CS 202: Computer Science II
Assignment 13: Inception
Linked List inside of a Linked List inside of a Linked List!

"An idea is like a virus, resilient, highly contagious. ... Once an idea has taken hold of the brain it's almost impossible to eradicate. An idea that is fully formed - fully understood - that sticks; right in there somewhere."
–Mr. Cobb (**INCEPTION, 2010**)

That is exactly how this assignment was created. The idea of having Linked Lists inside other Linked list was proposed by Dr. Grzegorz Chmaj as a possible final project. That was on a Thursday, at some point that night, I came across something related to Inception and so this project was born. I then spent the weekend developing it and here it is…

In this assignment you will develop a Doubly Linked List of Nodes. Each of these nodes may contain a link to the beginning of other linked lists along with a data field that contains a string representing what type of node it is.

To better understand the project I welcome you to watch the introductory video walking you through it:

https://www.youtube.com/watch?v=qzn_mBfnAX0 (Like & Subscribe!)

The basic linked list represents REALITY and each node contains the string REALITY in its data field. From there if we want to enter a dream we create another linked list and have one node point to it. The nodes of this new linked list contain the string representing the name of the dream in their data field. For example if we call the dream VAN then that is what they will contain.

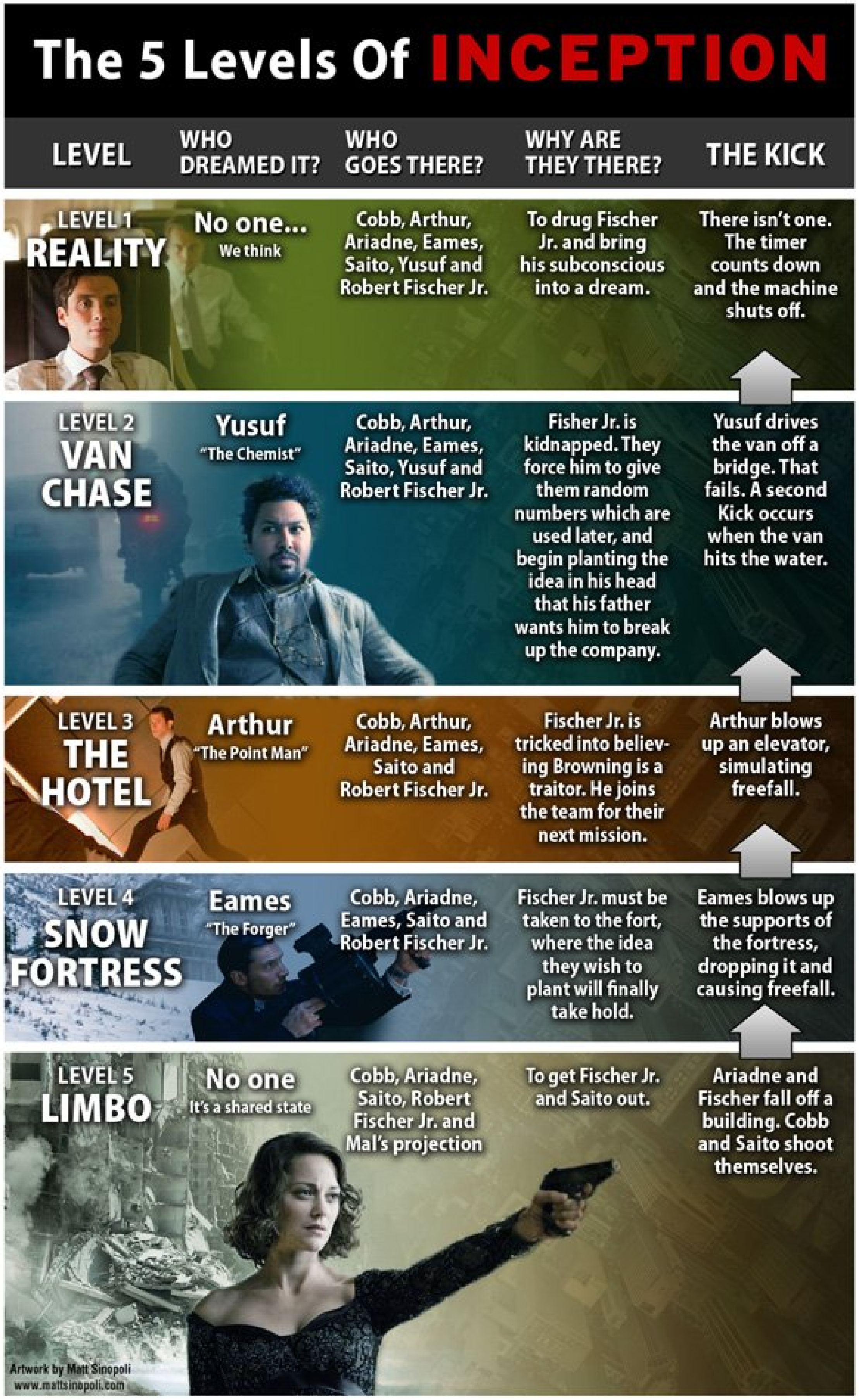
Next, just like the movie, if we enter a dream within a dream, then that is a linked list that is created in one of the dream nodes. So one of the VAN nodes would have a link to this second dream level which we can call HOTEL. This can go on and on with dreams within other dreams as deep as needed.

The program runs using linux (input) redirection to read in a file that contains on its first line all of the levels in the order from REALITY to the deepest level. Each must be separated by a space.

For example here is a sample input file with 5 levels (1 real, 4 dream layers)

```
REALITY VAN HOTEL FORTRESS LIMBO
REALITY REALITY REALITY REALITY REALITY REALITY REALITY REALITY REALITY REALITY
REALITY REALITY VAN VAN VAN VAN VAN VAN VAN VAN HOTEL HOTEL VAN HOTEL VAN HOTEL
FORTRESS HOTEL FORTRESS VAN FORTRESS HOTEL FORTRESS HOTEL FORTRESS FORTRESS
FORTRESS HOTEL FORTRESS LIMBO LIMBO FORTRESS LIMBO LIMBO VAN FORTRESS FORTRESS
LIMBO FORTRESS HOTEL VAN REALITY REALITY REALITY REALITY REALITY
```

The first line defines our universe and then the second line and on define the nodes of the linked list in that specific order. Notice that we go back and forth from VAN to HOTEL that just adds to the ending of each of the ongoing linked lists. The only difference from inserting them all in order is what ID each node gets since every node gets an ID representing the number of node it was when created (a counter increases every time a node is created). I found the following graphic that explains the movie INCEPTION along with what each of the levels in the input file represent. It was created by CinemaBlend.com: Source: https://www.cinemablend.com/new/An-Illustrated-Guide-5-Levels-Inception-19643.html

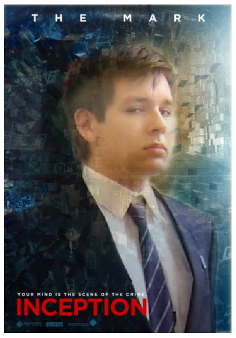


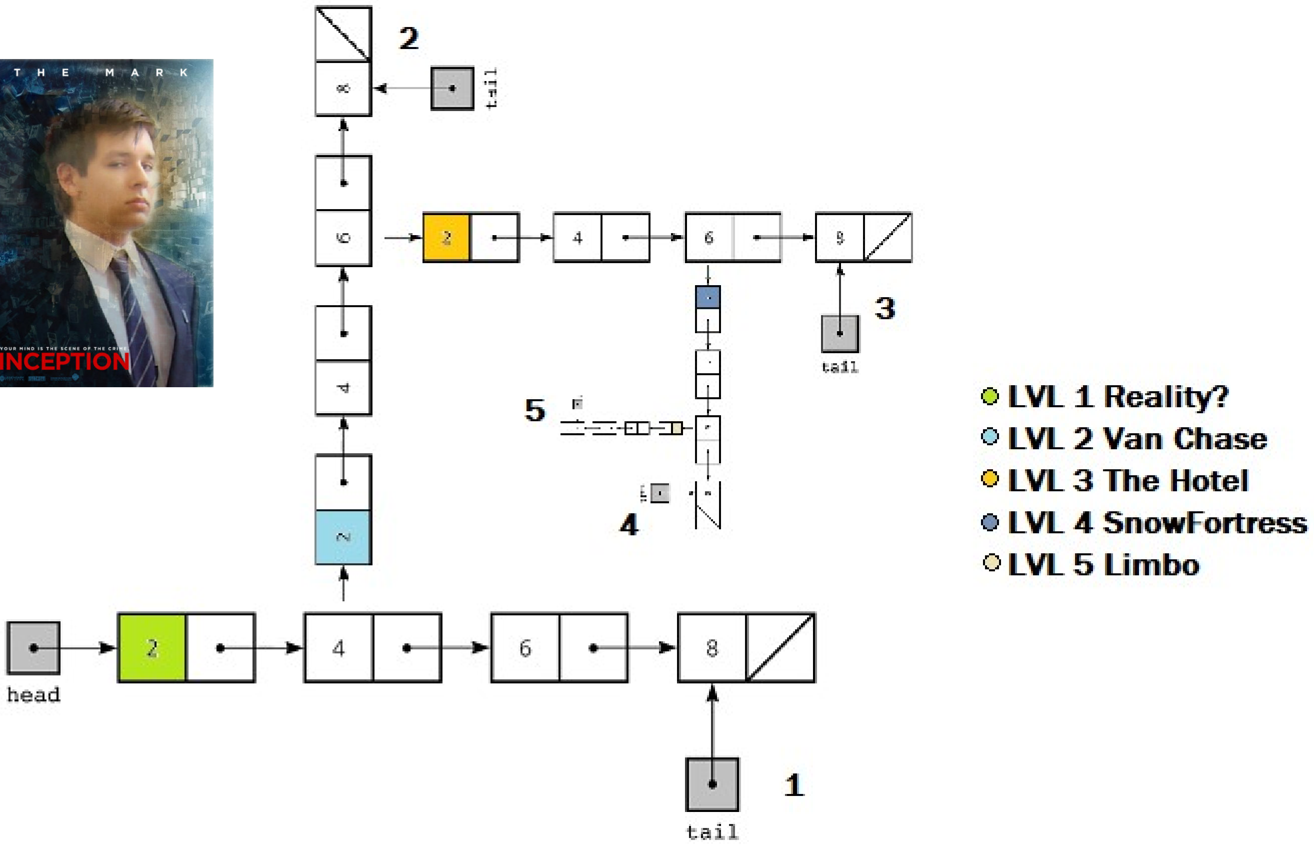
The 5 Levels Of INCEPTION

LEVEL	WHO DREAMED IT?	WHO GOES THERE?	WHY ARE THEY THERE?	THE KICK
LEVEL 1 **REALITY**	**No one...** We think	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	To drug Fischer Jr. and bring his subconscious into a dream.	There isn't one. The timer counts down and the machine shuts off.
LEVEL 2 **VAN CHASE**	**Yusuf** "The Chemist"	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	Fisher Jr. is kidnapped. They force him to give them random numbers which are used later, and begin planting the idea in his head that his father wants him to break up the company.	Yusuf drives the van off a bridge. That fails. A second Kick occurs when the van hits the water.
LEVEL 3 **THE HOTEL**	**Arthur** "The Point Man"	Cobb, Arthur, Ariadne, Eames, Saito and Robert Fischer Jr.	Fischer Jr. is tricked into believing Browning is a traitor. He joins the team for their next mission.	Arthur blows up an elevator, simulating freefall.
LEVEL 4 **SNOW FORTRESS**	**Eames** "The Forger"	Cobb, Ariadne, Eames, Saito and Robert Fischer Jr.	Fischer Jr. must be taken to the fort, where the idea they wish to plant will finally take hold.	Eames blows up the supports of the fortress, dropping it and causing freefall.
LEVEL 5 **LIMBO**	**No one** It's a shared state	Cobb, Ariadne, Saito, Robert Fischer Jr. and Mal's projection	To get Fischer Jr. and Saito out.	Ariadne and Fischer fall off a building. Cobb and Saito shoot themselves.

Artwork by Matt Sinopoli
www.mattsinopoli.com

The next graphic represents how these 5 levels will be stored in our program by each doubly linked list. Each node has a linked list pointer, but only the one that points to a dream level is not set to null.





Because it is a doubly linked list you can go back and forth in-between the same level/list's nodes but there is no way to jump out of the dream level you are at as there is no pointer in the beginning of the linked list that points back to the previous level. To get out you will need a kick, a.k.a to wake up from that dream! And because once you wake up from a dream you cannot go back into it as it would be a new dream. Once you wake up you de-allocate and lose access to that dream.

To create the program you will need the iostream, string and sstream library files. The sstream library comes in useful to read that first line of the file and then split it into each word so we can count how many levels to setup. Because we use string::to_string() at some point in the program you will need to compile using std=c++11 or higher.

In order to convey what each class should contain, here is the Unified Modeling Language Class Diagram for each class. I also threw in the friend functions that they have as well.

Remember that:
+ stands for public
− stands for private, and
# stands for protected.

First we have the Levels class.

Levels
+ name: string*   **+ size: int**
+ Levels(int s = 0)   **+ ~Levels()**   **+ findLevel(string search) : int //Returns the level it's searching**   **+ friend operator<<(ostream& os, const Levels& lvl) : ostream&**   **+ friend operator>>(istream& is, Levels& lvl) : istream&**

```
//The input file consists of all of the dream levels in the right order in the
//first line. The second line is all of the different ticks that identify which
//dream level they represent. See sample input files.

//Maintains the dream levels in the world. Class stores them in array of strings
//that we can use to check where in the dream hierarchy we are.
```

As you can see this class's job is to read in the levels from the first line of the input file, count them and create an array of them that stores their names. The stream insertion operator (<<) prints out "Level #: NAME" where # is the level number and NAME is the name of the level from the file. The stream extraction operator >> uses getline to read the first line of the input file, then stringstream to then use getline again on this sstream variable to read each individual word and count them. Your cefault constructor should not do anything, but as a fallback if you call the constructor with a number it will create levels and just name them after the number they are. This can be useful for testing purposes.

Next we have Node class, which is the heart of the linked lists.
Note: This requires a forward declaration of the LinkedList class to work. This can be achieved by typing: `class LinkedList; //Forward Declaration`

```
                              Node
+ data: string*    //contains what dream it is, or REALITY if not
+ DeeperDream: LinkedList* //points to a dream if there is one
+ next: Node*              //Points to next node
+ prev: Node*              //Points to previous node
- id: int                  //unique, gets set based on count
- count: static int        //Initialize to 0
+ Node()   //Sets everything to null, allocates a string for data
+ ~Node()  //delete data, set everything else to null
+ getID() const: int
+ friend operator<<(ostream& os, const Node& node) : ostream&
+ friend operator<<(ostream& os, const Node* node) : ostream&
+ friend operator>>(istream& is, Node& node) : istream&
+ friend operator>>(istream& is, Node* node) : istream&
```

```
//Nodes that make the heart of the linked list.
//Data is a string which identifies what level we are at.
//The DeeperDream is kept null unless this signifies the beginning
//of a deeper dream level.
//Because it is a doubly linked list, we have a prev and next pointer.
//Both the node and linked list class have a static to keep track of
//how many objects have been made so each can have a unique identifier
```

In addition to the above comments. Keep everything to null if its not pointing to anything to avoid issues. Your stream insertion operator should print "Node #: D" where # is the id and D is the contents of the data string. Your stream extraction operator should be a single line that reads into the data string. Note that the constructor copies the value of count into id and then increases it. **`id(count++)`**

Next we have the LinkedList class.

```
                           LinkedList
+ head: Node*              //Points to first node
+ tail: Node*              //Points to last node
- id: int                  //unique, gets set based on count
- count: static int        //Initialize to 0
+ LinkedList()  //Sets everything to Null, id(count++)
+ ~LinkedList() //De-allocates and Nulls all of the nodes in list
+ getID() const: int
+ friend operator<<(ostream& os, const LinkedList& node) : ostream&
+ friend operator<<(ostream& os, const LinkedList* node) : ostream&
+ fwdPrintList(): void  //Prints List from Head to Tail (use next)
+ bkwdPrintList(): void //Prints List from Tail to Head (use prev)
+ insertEndNode(Node *n): void
+ insertDreamNode(Node *n, Levels &lvl): void
+ gotoLevel(int lvl) : LinkedList* //Returns Pointer to beginning
              //of list at the required level passed in parameter
```

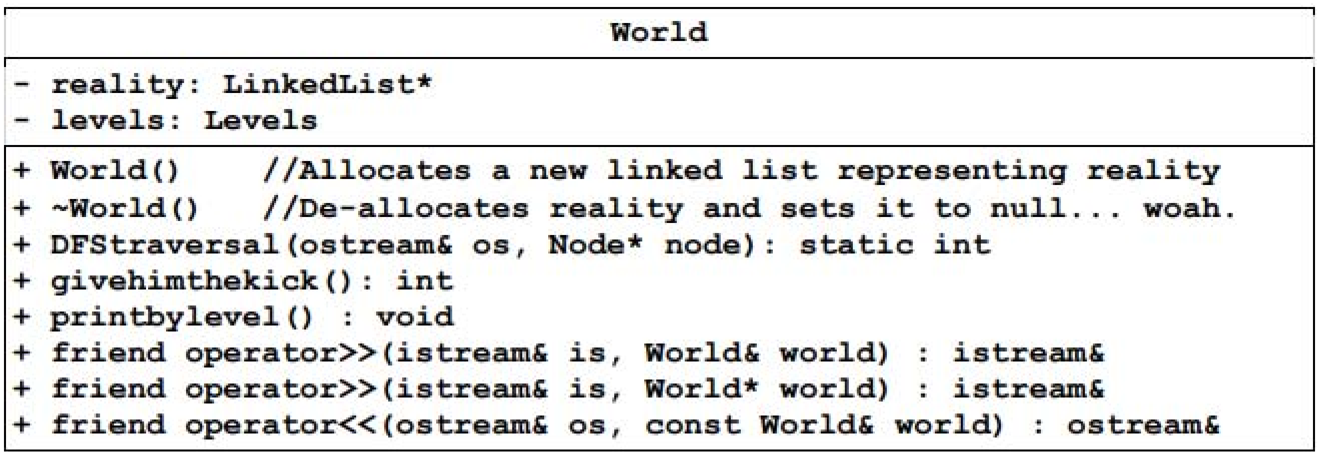
This class stores all of the nodes and has a head and tail that point to the beginning and end of the linked list. It is a doubly linked list. The `insertEndNode` function will insert at the end of the list (or at the beginning if it is the first item. Make sure to update the head and tail accordingly depending on where you insert.

`insertDreamNode` on the other hand inserts a dream node to the appropriate dream level so if it is a dream within a dream then it will insert at the required level. Note that levels are printed starting as level 1 but internally are kept starting at level 0. So level 0 will always be REALITY. To find out where it starts it calls the data member variable of the node parameter (n) and searches for that level in the Levels parameter (lvl). It does this using the `findlevel` member function of the Levels class.

Next to insert it needs to go to the level that we get from the function call. So for that we call the `gotoLevel` function that returns a pointer of LinkedList type that is the beginning of the required linked list where we have to insert. `gotoLevel` is probably the hardest function you will need to code as you have to allocate a new linked list if when inserting a node that dream level has not been created yet. You also need to error check if you try to enter a dream within a dream with a dream (2 levels deeper than you are) if there is no dream created. You can always create a dream where you are. Think of it like you can always go to sleep and start dreaming. Even if you are in a dream already, but you cannot go to sleep and dream, 2 layers in. First that dream has to start then you can sleep again and make second layer.

The stream insertion operator should print LinkedList: # where # is the id of the linked list.

Finally we have the World class.



```
World
- reality: LinkedList*
- levels: Levels
+ World()     //Allocates a new linked list representing reality
+ ~World()    //De-allocates reality and sets it to null... woah.
+ DFStraversal(ostream& os, Node* node): static int
+ givehimthekick(): int
+ printbylevel() : void
+ friend operator>>(istream& is, World& world) : istream&
+ friend operator>>(istream& is, World* world) : istream&
+ friend operator<<(ostream& os, const World& world) : ostream&
```

The World class completes the OOD approach by encapsulating every other object into one nice package. The World class has both a LinkedList pointer called reality to get started building the first level and a Levels variable to store the level information read from the file. From there we have a few functions. `DFStraversal` is a recursive static function that takes a node pointer (which should initially be called on the first node of reality or wherever we want to start from. It will then recursively call itself as it traverses and prints all of the levels in a Depth First Search manner. Meaning it will try to go as deep as possible (printing as it visit each node) until it cannot go further. Then it backtracks to where it has visited (it only prints the first time it reaches a new node) and then once it comes back a dream level it continues that level to finish it then backtracks to the previous level and does the same. Review the sample output to better understand this algorithm. At the end it also prints out how many nodes it visited as the running total (hence why its value returning). This function gets called by the overloaded stream insertion operator.

The `printbylevel` function calls the `fwdPrintList` function on each level for a different way to represent the architecture.

The `givehimthekick` function is how you de-allocate all the dream layers. Note that the reality layer is not de-allocated here, that's in the World destructor (fitting name). The way this works is first it creates a one dimensional array of LinkedList pointers then it goes through the architecture and gets a pointer to all the layers (so all the linked lists. Then it goes through and de-allocates all lists at once. You can think of this as a synchronized kick to wake up no matter how many layers deep you are!

Finally, the stream extraction operator (>>) is overloaded so that it can read in all of the input from the file by calling the `insertEndNode` or the `insertDreamNode` depending on what it is inserting. This is done by cross checking the word read in to what level it belongs to. It prints out REALITY when inserting a reality node and it prints out DREAM when inserting a dream node regardless of which layer it is working on

You are given the following client code that you are not allowed to modify:

```
int main(int, char**){/////////
    cout << "Inception" << endl << endl;//
    World *world = new World;/////////////
    cin >> world->levels;/////////////////
    //cout << world->levels;//////////////
    cin >> world;/////////////////////////
    //world->reality->fwdPrintList();/////
    //world->reality->bkwdPrintList();////
    world->printbylevel();////////////////
    cout << *world;///////////////////////
    world->givehimthekick();//////////////
    delete world; world = NULL;///////////
    cout << "\nTHE END\n\nCreated by ";///
    cout << "Jorge Fonseca" << endl;//////
    return 0;/////////////////////////////
}   //////////////////////////////////////
```

Your operator overloading should work so that the output matches the given samples. Once you have finished your class designs and implementations

Remember, you must use linux input redirection. Meaning that given the file: `InceptionMission1`, and your executable file `a.out`, your program should execute as follows:

```
./a.out < InceptionMission2
```

With the output printing to the terminal. This is not filestreams. Do not use filestreams.

Notes:
-If the members are "read-only" put the appropriate const tag. The same applies to parameters passed by reference that are not modified in a function. It is a good practice that you should get used to doing.
-Make sure you name your program file in accordance with the syllabus stipulations
-Test your program by running it a few times with different method calls and inputs to make sure the output is correct for all cases. My sample client code and test files are by no means enough to fully test all of your program's edge cases. Come up with your own! This is a good skill to learn.
-Make sure your output is EXACTLY formatted like the given sample outputs or you will lose points. This includes spacing, line feeds, and even upper/lower casing of words. Use linux's **diff** command to compare outputs so that you can find any potential issues.
-Comment your source code appropriately according to the stipulations on the rubric. Everything including skeleton code, if given, should be documented.
-Even though we have covered multifile compilation. Please make this program in one .cpp file
-If you are given skeleton code, then you are not allowed to modify my code! So if something is not working, it is because you need to change what you wrote.
-**Your program must have 0 errors from 0 contexts, and 0 bytes in 0 blocks definitely, indirectly, or possibly lost in valgrind.** That means **NO MEMORY LEAKS** allowed.

Good luck.

Attached are four input and output files along with the valgrind output for each:
Note: All four valgrind outputs are found in the same file: `valgrind-outputs.txt`
The input and output files are as follows:
Input: **InceptionMission0**, Output: **output0.txt**
Input: **InceptionMission1**, Output: **output1.txt**
Input: **InceptionMission2**, Output: **output2.txt**
Input: **InceptionMissionCrash**, Output: **outputCrash.txt**

For your convenience here is Input: **InceptionMission2**, Output: **output2.txt** originally executed as:

```
./a.out < InceptionMission2
```

This is the input/output that matches the graphs shown earlier.

Input File:

```
REALITY VAN HOTEL FORTRESS LIMBO
REALITY REALITY REALITY REALITY REALITY REALITY REALITY REALITY REALITY REALITY
REALITY REALITY VAN VAN VAN VAN VAN VAN VAN VAN HOTEL HOTEL VAN HOTEL VAN HOTEL
FORTRESS HOTEL FORTRESS VAN FORTRESS HOTEL FORTRESS HOTEL FORTRESS FORTRESS
FORTRESS HOTEL FORTRESS LIMBO LIMBO FORTRESS LIMBO LIMBO VAN FORTRESS FORTRESS
LIMBO FORTRESS HOTEL VAN REALITY REALITY REALITY REALITY REALITY
```

Required output to Terminal:

```
$ ./a.out < InceptionMission2
Inception

New Linked List
5 Levels Created.
New Node: REALITY
New Node: REALITY
New Node: REALITY
New Node: REALITY
New Node: REALITY
New Node: REALITY
New Node: REALITY
New Node: REALITY
New Node: REALITY
New Node: REALITY
New Node: REALITY
New Node: REALITY
New Node: DREAM
Searching LVL:2
New Linked List
We need to go deeper: 0 1
Inserting to LVL2: VAN
New Node: DREAM
Searching LVL:2
We are going deeper: 0 1
FOUND?: VAN
Inserting to LVL2: VAN
New Node: DREAM
Searching LVL:2
We are going deeper: 0 1
FOUND?: VAN
Inserting to LVL2: VAN
New Node: DREAM
Searching LVL:2
We are going deeper: 0 1
FOUND?: VAN
Inserting to LVL2: VAN
New Node: DREAM
Searching LVL:2
We are going deeper: 0 1
FOUND?: VAN
Inserting to LVL2: VAN
New Node: DREAM
Searching LVL:2
We are going deeper: 0 1
FOUND?: VAN
Inserting to LVL2: VAN
New Node: DREAM
Searching LVL:2
We are going deeper: 0 1
FOUND?: VAN
Inserting to LVL2: VAN
New Node: DREAM
Searching LVL:2
We are going deeper: 0 1
FOUND?: VAN
Inserting to LVL2: VAN
New Node: DREAM
Searching LVL:3
We are going deeper: 0 2
New Linked List
We need to go deeper: 1 2
Inserting to LVL3: HOTEL
New Node: DREAM
Searching LVL:3
We are going deeper: 0 2
We are going deeper: 1 2
FOUND?: HOTEL
Inserting to LVL3: HOTEL
New Node: DREAM
Searching LVL:2
We are going deeper: 0 1
FOUND?: VAN
Inserting to LVL2: VAN
New Node: DREAM
Searching LVL:3
We are going deeper: 0 2
We are going deeper: 1 2
FOUND?: HOTEL
Inserting to LVL3: HOTEL
New Node: DREAM
Searching LVL:2
We are going deeper: 0 1
FOUND?: VAN
Inserting to LVL2: VAN
New Node: DREAM
Searching LVL:3
We are going deeper: 0 2
We are going deeper: 1 2
FOUND?: HOTEL
Inserting to LVL3: HOTEL
New Node: DREAM
Searching LVL:4
We are going deeper: 0 3
We are going deeper: 1 3
New Linked List
We need to go deeper: 2 3
Inserting to LVL4: FORTRESS
New Node: DREAM
Searching LVL:3
We are going deeper: 0 2
We are going deeper: 1 2
FOUND?: HOTEL
Inserting to LVL3: HOTEL
New Node: DREAM
Searching LVL:4
We are going deeper: 0 3
We are going deeper: 1 3
We are going deeper: 2 3
FOUND?: FORTRESS
Inserting to LVL4: FORTRESS
New Node: DREAM
Searching LVL:2
We are going deeper: 0 1
FOUND?: VAN
Inserting to LVL2: VAN
New Node: DREAM
Searching LVL:4
We are going deeper: 0 3
We are going deeper: 1 3
We are going deeper: 2 3
FOUND?: FORTRESS
Inserting to LVL4: FORTRESS
New Node: DREAM
Searching LVL:3
```

```
We are going deeper: 0 2
We are going deeper: 1 2
FOUND?: HOTEL
Inserting to LVL3: HOTEL
New Node: DREAM
Searching LVL:4
We are going deeper: 0 3
We are going deeper: 1 3
We are going deeper: 2 3
FOUND?: FORTRESS
Inserting to LVL4: FORTRESS
New Node: DREAM
Searching LVL:3
We are going deeper: 0 2
We are going deeper: 1 2
FOUND?: HOTEL
Inserting to LVL3: HOTEL
New Node: DREAM
Searching LVL:4
We are going deeper: 0 3
We are going deeper: 1 3
We are going deeper: 2 3
FOUND?: FORTRESS
Inserting to LVL4: FORTRESS
New Node: DREAM
Searching LVL:4
We are going deeper: 0 3
We are going deeper: 1 3
We are going deeper: 2 3
FOUND?: FORTRESS
Inserting to LVL4: FORTRESS
New Node: DREAM
Searching LVL:4
We are going deeper: 0 3
We are going deeper: 1 3
We are going deeper: 2 3
FOUND?: FORTRESS
Inserting to LVL4: FORTRESS
New Node: DREAM
Searching LVL:3
We are going deeper: 0 2
We are going deeper: 1 2
FOUND?: HOTEL
Inserting to LVL3: HOTEL
New Node: DREAM
Searching LVL:4
We are going deeper: 0 3
We are going deeper: 1 3
We are going deeper: 2 3
FOUND?: FORTRESS
Inserting to LVL4: FORTRESS
New Node: DREAM
Searching LVL:5
We are going deeper: 0 4
We are going deeper: 1 4
We are going deeper: 2 4
New Linked List
We need to go deeper: 3 4
Inserting to LVL5: LIMBO
```

```
New Node: DREAM
Searching LVL:5
We are going deeper: 0 4
We are going deeper: 1 4
We are going deeper: 2 4
We are going deeper: 3 4
FOUND?: LIMBO
Inserting to LVL5: LIMBO
New Node: DREAM
Searching LVL:4
We are going deeper: 0 3
We are going deeper: 1 3
We are going deeper: 2 3
FOUND?: FORTRESS
Inserting to LVL4: FORTRESS
New Node: DREAM
Searching LVL:5
We are going deeper: 0 4
We are going deeper: 1 4
We are going deeper: 2 4
We are going deeper: 3 4
FOUND?: LIMBO
Inserting to LVL5: LIMBO
New Node: DREAM
Searching LVL:5
We are going deeper: 0 4
We are going deeper: 1 4
We are going deeper: 2 4
We are going deeper: 3 4
FOUND?: LIMBO
Inserting to LVL5: LIMBO
New Node: DREAM
Searching LVL:2
We are going deeper: 0 1
FOUND?: VAN
Inserting to LVL2: VAN
New Node: DREAM
Searching LVL:4
We are going deeper: 0 3
We are going deeper: 1 3
We are going deeper: 2 3
FOUND?: FORTRESS
Inserting to LVL4: FORTRESS
New Node: DREAM
Searching LVL:4
We are going deeper: 0 3
We are going deeper: 1 3
We are going deeper: 2 3
FOUND?: FORTRESS
Inserting to LVL4: FORTRESS
New Node: DREAM
Searching LVL:5
We are going deeper: 0 4
We are going deeper: 1 4
We are going deeper: 2 4
We are going deeper: 3 4
FOUND?: LIMBO
Inserting to LVL5: LIMBO
New Node: DREAM
```

```
Searching LVL:4
We are going deeper: 0 3
We are going deeper: 1 3
We are going deeper: 2 3
FOUND?: FORTRESS
Inserting to LVL4: FORTRESS
New Node: DREAM
Searching LVL:3
We are going deeper: 0 2
We are going deeper: 1 2
FOUND?: HOTEL
Inserting to LVL3: HOTEL
New Node: DREAM
Searching LVL:2
We are going deeper: 0 1
FOUND?: VAN
Inserting to LVL2: VAN
New Node: REALITY
New Node: REALITY
New Node: REALITY
New Node: REALITY
New Node: REALITY
New Node:
Printing World by Level:

Forward Print List, REALITY Level:
Node 0: REALITY, Node 1: REALITY, Node 2: REALITY, Node 3: REALITY, Node 4:
REALITY, Node 5: REALITY, Node 6: REALITY, Node 7: REALITY, Node 8: REALITY, Node
9: REALITY, Node 10: REALITY, Node 11: REALITY, Node 51: REALITY, Node 52: REALITY,
Node 53: REALITY, Node 54: REALITY, Node 55: REALITY, ▯▯.

Forward Print List, VAN Level:
Node 12: VAN, Node 13: VAN, Node 14: VAN, Node 15: VAN, Node 16: VAN, Node 17: VAN,
Node 18: VAN, Node 19: VAN, Node 22: VAN, Node 24: VAN, Node 29: VAN, Node 44: VAN,
Node 50: VAN, ▯▯.

Forward Print List, HOTEL Level:
Node 20: HOTEL, Node 21: HOTEL, Node 23: HOTEL, Node 25: HOTEL, Node 27: HOTEL,
Node 31: HOTEL, Node 33: HOTEL, Node 37: HOTEL, Node 49: HOTEL, ▯▯.

Forward Print List, FORTRESS Level:
Node 26: FORTRESS, Node 28: FORTRESS, Node 30: FORTRESS, Node 32: FORTRESS, Node
34: FORTRESS, Node 35: FORTRESS, Node 36: FORTRESS, Node 38: FORTRESS, Node 41:
FORTRESS, Node 45: FORTRESS, Node 46: FORTRESS, Node 48: FORTRESS, ▯▯.

Forward Print List, LIMBO Level:
Node 39: LIMBO, Node 40: LIMBO, Node 42: LIMBO, Node 43: LIMBO, Node 47: LIMBO, ▯▯.

The World As We Know It:
Level 1: REALITY
Level 2: VAN
Level 3: HOTEL
Level 4: FORTRESS
Level 5: LIMBO

Dream Architecture:
0REALITY, 1REALITY, 2REALITY, 3REALITY, 4REALITY, 5REALITY, 6REALITY, 7REALITY,
8REALITY, 9REALITY, 10REALITY, 11REALITY, ENTERING DEEPER LAYER, 12VAN, 13VAN,
14VAN, 15VAN, 16VAN, 17VAN, 18VAN, 19VAN, ENTERING DEEPER LAYER, 20HOTEL, 21HOTEL,
23HOTEL, 25HOTEL, ENTERING DEEPER LAYER, 26FORTRESS, 28FORTRESS, 30FORTRESS,
32FORTRESS, 34FORTRESS, 35FORTRESS, 36FORTRESS, 38FORTRESS, ENTERING DEEPER LAYER,
39LIMBO, 40LIMBO, 42LIMBO, 43LIMBO, 47LIMBO, EXITING A LAYER, 41FORTRESS,
45FORTRESS, 46FORTRESS, 48FORTRESS, EXITING A LAYER, 27HOTEL, 31HOTEL, 33HOTEL,
37HOTEL, 49HOTEL, EXITING A LAYER, 22VAN, 24VAN, 29VAN, 44VAN, 50VAN, EXITING A
LAYER, 51REALITY, 52REALITY, 53REALITY, 54REALITY, 55REALITY, ▯▯.
Node Visited: 56

Non, rien de rien, Non, je ne regrette rien, Ni le bien qu'on m'a fait BRRRR
BRRRRRRR BRRRRRR BRRRRRRRRR
THE DREAM IS COLLAPSING!
THE DREAM IS COLLAPSING!
THE DREAM IS COLLAPSING!
THE DREAM IS COLLAPSING!

THE END

Created by Jorge Fonseca
```

Inception (2010), directed by Christopher Nolan.
© 2010 Warner Brothers Entertainment Company; all rights reserved