CS 3100 – *Data Structures and Algorithms*

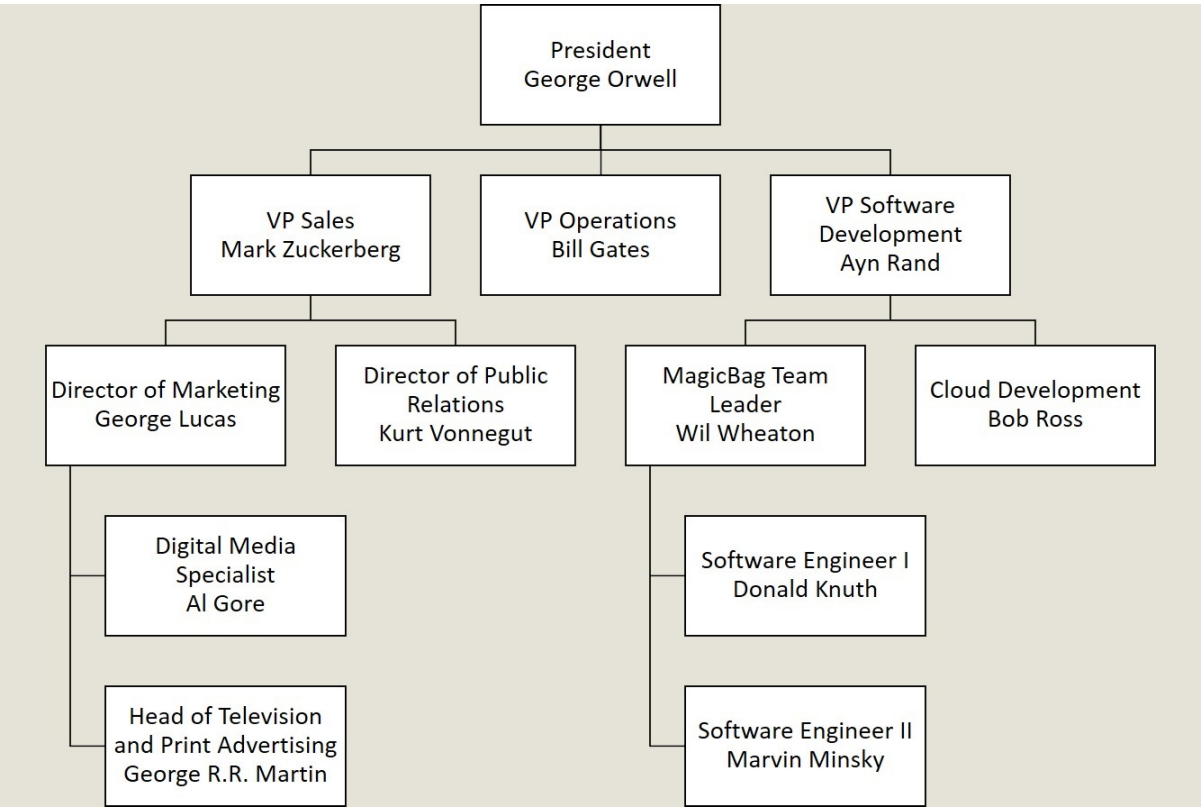
Project #2 - Managing a Hierarchy using a General Tree

Learning Objectives

- Apply object-oriented programming concepts in C++
- Choose and implement an appropriate internal representation for a specified abstract data type
- Design, implement, and use a general tree data structure
- Analyze operations for time complexity

Overview

Your task for this assignment is to implement a general tree that can store, manipulate, and query the organizational chart for a company. The organizational chart contains the title and full name of every employee in the company. The structure of the chart shows who works for whom within the company. For example, the following organizational chart might represent the fictional company *Magic Bags, Incorporated (MBI)*:



The Organization Tree

You will store the organization chart as a general tree. You may choose any of the general tree implementations we have discussed in class. Your tree should satisfy the following specifications:

- So that we can test your code, we need to know if you will use a TreeNode* (for linked implementations) or an integer (for array implementations) to point to a TreeNode. In your OrgTree.h file, you should use one of the following two code blocks:

For array-based implementations, you should use:

```
#define TREENODEPTR int
#define TREENULLPTR -1
```

For linked node implementations, you should use:

```
#define TREENODEPTR TreeNode*
#define TREENULLPTR NULL
```

- Each tree node should store a title and a name. You can use the C++ string class for both of these data elements. (Don't forget to **`#include <string>`**)

- **`void OrgTree::addRoot(title, name)`** – Creates a root node for the tree. If the tree already has a root, the entire tree becomes a subtree of the new root.

- **`unsigned int OrgTree::getSize()`** – Returns the number of employees in the tree.

- **`TREENODEPTR OrgTree::getRoot()`** – Returns a pointer to the root node in the tree. Depending on your implementation, this function may return a TreeNode* or an integer that will be used as an array index.

- **`TREENODEPTR OrgTree::leftmostChild(TREENODEPTR node)`** – Returns a pointer to the leftmost child of the node passed to the function. If the node has no children, returns TREENULLPTR.

- **`TREENODEPTR OrgTree::rightSibling(TREENODEPTR node)`** – Returns a pointer to the right sibling of the node passed to the function. If the node has no right sibling, returns TREENULLPTR.

- **`void OrgTree::printSubTree(TREENODEPTR subTreeRoot)`** – print out the subtree starting at the node pointed to by subTreeRoot. This function should use indentation to show the tree structure. For example, if you called printSubTree on the tree shown above, and passed it a pointer to the "VP Sales" node, you would see the following:

```
VP Sales: Mark Zuckerberg
        Director of Marketing: George Lucas
                Digital Media Specialist: Al Gore
                Head of Television and Print Advertising: George R.R. Martin
        Director of Public Relations: Kurt Vonnegut
```

- **`TREENODEPTR OrgTree::find(title)`** – returns a TREENODEPTR to the node listing the given title (exact string match with the title string in a TreeNode object). If the title is not found, the function should return

TREENULLPTR. For simplicity, you may assume that all titles in the company are unique.

- **`bool OrgTree::read(filename)`** – reads an organization tree from a file. The file will contain one tree node per line, except for lines containing only ')' characters, which mark the ends of subtrees. The organization tree illustrated above would be represented by the file shown below. If the file is found and read successfully, this function should return **true**. If the file is not found or the file is improperly formatted, the function should return **false**.

```
President, George Orwell
VP Sales, Mark Zuckerberg
Director of Marketing, George Lucas
Digital Media Specialist, Al Gore
)
Head of Television and Print Advertising, George Martin
)
)
Director of Public Relations, Kurt Vonnegut
)
)
VP Operations, Bill Gates
)
VP Software Development, Ayn Rand
MagicBag Team Leader, Wil Wheaton
Software Engineer I, Donald Knuth
)
Software Engineer II, Marvin Minsky
)
)
Cloud Development, Bob Ross
)
)
)
```

- **`void OrgTree::write(filename)`** – write out the OrgTree to a file, using the same file format described in the read() function above.

- **`void OrgTree::hire(TREENODEPTR, newTitle, newName)`** – Hire an employee. The employee should be added to the tree as the last child of the node pointed to by TREENODEPTR.

- **`bool OrgTree::fire(formerTitle)`** – Fire the employee who's title matches the argument to this function. If the title is found the employee's node in the tree is deleted and the function returns **true**. All employees of the fired employee now work directly for the fired employee's boss (e.g. all children of the deleted node become children of the deleted node's parent). If no match is found the function returns **false**. For simplicity, you can assume that titles in the company are unique. Note that you cannot fire the root node. If the formerTitle matches the root node, the function should return **false**.

- Your tree should be able to hold any number of employees. If you use an array-based tree implementation, your array must re-size when full.

Analysis and Documentation

- Each class and function should have appropriate header documentation/comments
- Each of the functions specified above must include the asymptotic run time, in Θ(*n*) notation, for a tree of size *n*, in the header comment block of the function.
- The OrgTree class comment block should include the space requirement for a tree of *n* employees.

Turn in and Grading

- Your OrgTree class should be implemented inline in the file OrgTree.h.
- If you define other classes, (such as TreeNode) you may use a separate, inline class, or a class defined entirely within Class OrgTree.
- Zip your entire project directory and turn it in via pilot. Make sure that all your files are included. Remember that code that will not compile will receive a grade of zero.

This Project is worth 50 points, distributed as follows:

Task	Points
`addRoot()` functions correctly and creates a valid OrgTree object	1
`getSize()` correctly returns the size for any OrgTree object	2
`getRoot()` returns a pointer to the root of the tree	2
`leftmostChild()` returns a pointer to the leftmost child of a node, or TREENULLPTR for nodes with no children	2
`rightSibling()` returns a pointer to the right sibling of a node, or TREENULLPTR for nodes with no right sibling	2
`printSubTree()` correctly prints the appropriate subtree, with correct indentation	3
`find()` correctly finds the node matching the given title, or returns TREENULLPTR if the title is not found in the tree	3
`read()` correctly reads and creates a tree from a properly formatted data file, or returns **false** if the file is not found or incorrectly formatted	5
`write()` correctly writes the tree to a properly formatted data file	5
`hire()` correctly inserts a new node into the tree	5
`fire()` correctly deletes a node from the tree, moving all employees to the parent node, or returns **false** if the formerTitle is not found or matches the root node of the tree	5
The tree can store any number of employees, subject to the amount of RAM available to the program	5
Your code is well organized, clearly written, and well-documented	5

The asymptotic run time for each member function and the space requirements of your tree are included in the documentation.	5