CSCI 2270 – Data Structures

Instructor: Asa Ashraf

Assignment 7 - Binary Search Trees II

Due Sunday, October 24th 2021, 11:59 PM

OBJECTIVES

1. **Create a super data structure combining BST and LL**
2. **Left Rotate BST**

Overview

Your team has received a revised specification for the Employee Directory application you developed in Assignment 6. In this assignment, you will tackle this new specification.

Conceptually, this assignment builds on top of the previous assignment but with a different data structure. Please make sure the header files are downloaded from the current assignment and not the previous one. *DO NOT MODIFY THE HEADER FILE. However, you may implement helper functions in your .cpp file.*

EmployeeDirectory Class

Your task is to implement a Binary Search Tree (BST) where each node contains an associated Linked List (LL) of employees. You are also required to perform a left rotation of a given node while maintaining the BST property.

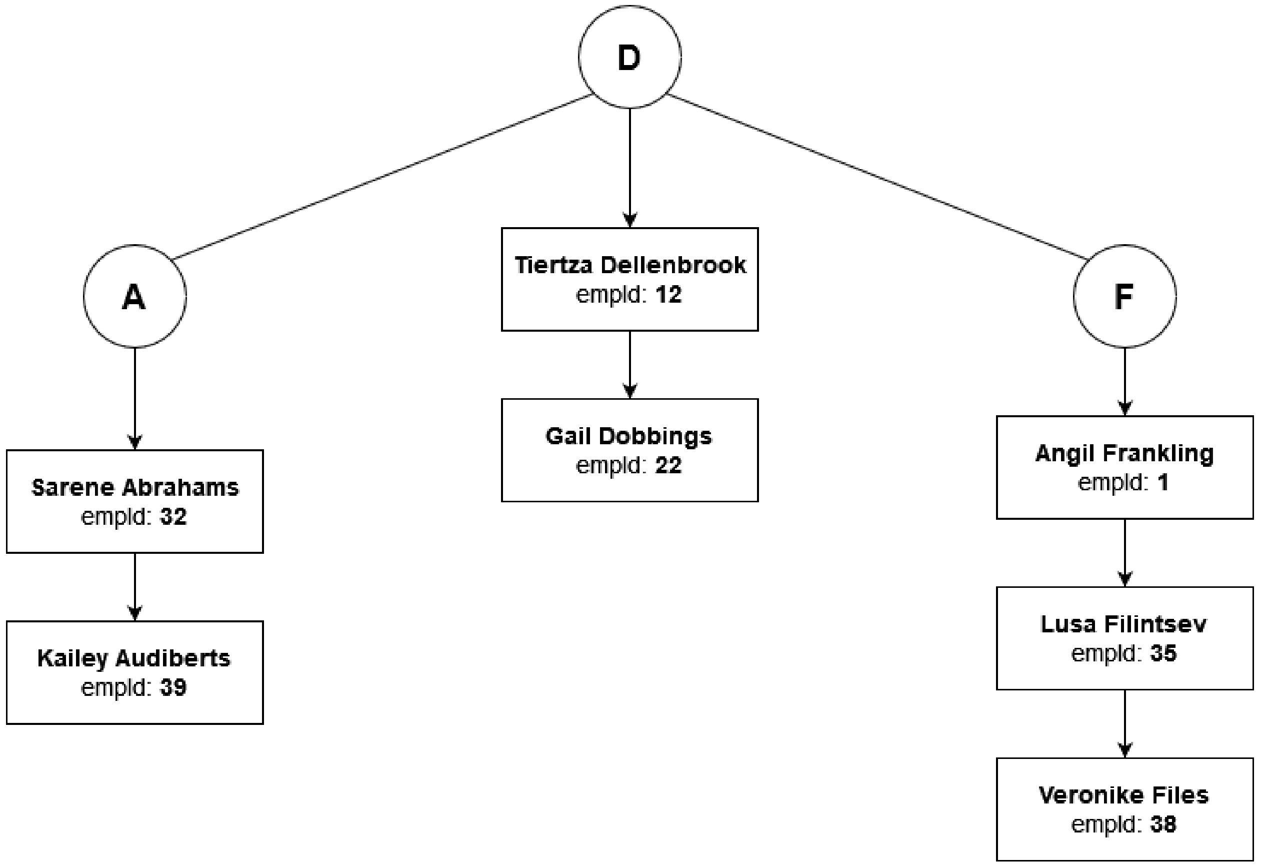
The key of a BST node is the last initial (first letter of the last name, *upper-case*) of the employees' name. The nodes of the tree are thus alphabetically ordered by this key i.e., with '**D**' as a parent node '**C**' appears in its left sub-tree and '**E**' appears in its right subtree.

Since multiple employees may share their last initial, their details will be stored in a linked list, which is sorted in the ascending order of their `empId`, under the appropriate BST node. Note that `empId` is unique. You may assume that each employee's empName is in the format: "`<first_name> <last_name>`", and that the initials are always upper-case letters.

You should also make note of the `parent` pointer in the BST nodes. For each node, this should refer to their correct parent in the tree, when the insertion is done. For the root node, this will be NULL.

You have been provided with parameterized constructors to help you allocate memory for the structs, as well as a couple of helper functions. A driver, similar to the one in Assignment 6, has also been provided. Please refer to the starter code, available on Canvas.

An example tree has been provided below:



The specification details of the class are as follows:

`EmployeeDirectory::EmployeeDirectory()`

- ➔ Constructor: Initialize `root = NULL`.

`EmployeeDirectory::~EmployeeDirectory()`

- ➔ Destructor: Free all memory that was allocated. This includes deallocating the memory of the underlying BST nodes' linked list as well.
- ➔ Autograder will check for any leaked memory in the final implementation check.

`TreeNode* EmployeeDirectory::searchCharNode(char key)`

- ➔ Search for and return the **`TreeNode t`**, where **`t->lastInitial==key`**.
- ➔ Utilize the BST structure to perform search.
- ➔ If a **`TreeNode`** does not exist with the given key, then return NULL.

`void EmployeeDirectory::showEmployeeDirectory()`

➔ Print the directory in the alphabetical order of the last initial. You should also print the BST node's parent key. Employee details of employees sharing the same last initial should be printed in the ascending order of their **`empId`**.

➔ For each **`TreeNode t`** you come across, use the below format. For the root node, print **`parent's lastInitial`** as **`' '`** (space character):

```
// for every non-empty TreeNode (t) in the tree:
// parent_lastInitial corresponds to the BST parent's lastInitial
cout << "[Employees with lastInitial=" << t->lastInitial << ",
parent=" << m->parent->lastInitial << "]" << endl;
```

➔ If a TreeNode contains an empty linked list, print the following for its linked list.

```
cout << "No employees found." << endl;
```

➔ Otherwise print the list using the following format:

```
// otherwise, for every LLNode (m) attached to t
cout << "> " << m->empName << ", " << m->empPhone << ", " << m->empId
<< endl;
```

➔ After you are done processing the linked list, print the following dashed line:

```
cout << "---------------------------------------" << endl;
```

<u>Sample Output for the example tree</u>

```
[Employees with lastInitial=A, parent=D]
> Sarene Abrahams, 265-620-8765, 8
> Kailey Audibert, 606-322-4324, 39
---------------------------------------
[Employees with lastInitial=D, parent= ]
> Tiertza Dellenbrook, 414-792-7435, 12
> Gail Dobbings, 477-684-7554, 22
---------------------------------------
[Employees with lastInitial=F, parent=D]
> Angil Frankling, 557-949-1496, 1
> Lusa Filintsev, 215-135-4990, 35
> Veronike Files, 865-213-6561, 38
---------------------------------------
```

`LLNode* EmployeeDirectory::searchEmployee(int empId, string empName)`

- ➔ Search for and return the `LLNode*` corresponding to `empId` and `empName`.
  - ◆ You will find the already implemented helper `getLastInitial(empName)` to be useful to extract the lastInitial character.
  - ◆ If you have implemented the `searchCharNode`, you may find it useful to get the pointer to the correct `TreeNode`.
  - ◆ Once you have the correct `TreeNode`, search the associated Linked List for the `LLNode`. If the node exists, return the pointer to the node. Else return `NULL`.
  - ◆ You may assume that while two employees may share the same name, no two employees have the same empId.

`void EmployeeDirectory::insertEmployee(int empId, string empName, int empLevel, string empPhone, int empJoiningYear)`

- ➔ Add employee to the data structure in the correct place based on the class description.
  - ◆ You will be inserting a new `LLNode` with the associated data (`empId, empName, empLevel, empPhone and empJoiningYear`).
  - ◆ If no tree node exists (based on the lastInitial), create a new tree node and insert the newly created Linked List node into it.
  - ◆ Otherwise, insert into an existing TreeNode's Linked List, such that the list is in ascending order of `empId`.
  - ◆ You may assume that no two employees have the same `empId`.
- ➔ **When adding a new `TreeNode`, ensure its parent pointers are correctly set.**

- ➔ **Example:** In the earlier figure, if we want to insert [*empName*: "Nicolas Flamel", *empId*: 5] - then it will be inserted on the TreeNode '**F**', between the nodes [*empName*: "Angil Frankling", *empId*: 1] and [*empName*: "Lusa Filintsev", *empId*: 35] because the new *empId*: 5 is between 1 and 35.

`void EmployeeDirectory::removeEmployee(string empName, int empId)`

- ➔ Delete the LLNode that contains the `empId`.
  - ◆ If deletion results in an empty Linked List, you should set the `head` to NULL. Do not delete the TreeNode.
  - ◆ If the employee does not exist in the data structure, print the following message

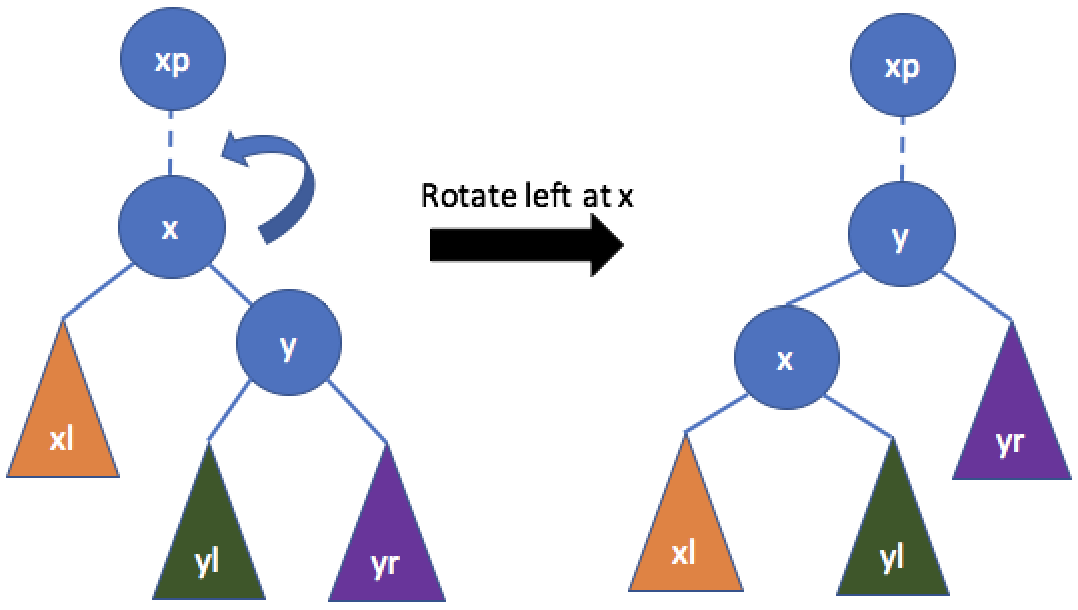
```
cout << "Employee not found."<< endl;
```

  - ◆ You may find `getLastInitial` and `searchCharNode` useful.
  - ◆ Please note that we use empName only for finding the BST node and then utilize empId to find and delete the LLNode.

`void EmployeeDirectory::leftRotate(TreeNode* curr)`

Rotate the node `curr` towards the left. Refer to the following illustration. A left rotation is performed at node **x.**

- ➔ Set the parent pointers accordingly:
    - ◆ Parent of `x` becomes the parent of `y`
    - ◆ Parent of `x` becomes `y`.
- ➔ Set the subtree (left and right children) pointers accordingly.
    - ◆ The left subtree of `y` becomes the right subtree of `x`.
    - ◆ `x` and its descendants become the left subtree of `y`.
- ➔ If `x` was the left (or right) subtree of `xp`, make `y` the left (or right) subtree of `x` respectively. This can be checked by comparing the title characters of the parent and the child's node.
- ➔ Ensure boundary conditions are accounted for:
    - ◆ `X` is root.
    - ◆ `X` has no right child.



The following functions are available as a part of the starter code.

`void EmployeeTree::getLastInitial(string name)`
This function returns the initial of the last name in the given argument. Assumes that the name is composed of "<first_name> <last_name>".

`void EmployeeTree::printEmployee(LLNode& n)`
This function prints the details of the given LLnode n.

Driver

Code for the driver (main function) is given as part of the starter code. Here is a brief description of the main function.

Your main function should first read information about each employee from a file and store that information in an `EmployeeDirectory` object. The name of the file is passed as a command-line argument. An example file is *Employees.csv* on Canvas. It is in the format:

```
<1:empId>,<1:empName>,<1:empLevel>,<1:empPhone>,<1:empJoiningYear>
<2:empId>,<2:empName>,<2:empLevel>,<2:empPhone>,<2:empJoiningYear>
```

We insert the nodes to the tree in the order they are read in. **After** reading in the information on each employee from the file, display a menu to the user.

```
cout << "======Main Menu======" << endl;
cout << "1. Find employee" << endl;
cout << "2. Insert employee" << endl;
cout << "3. Remove employee" << endl;
cout << "4. Show employee directory" << endl;
cout << "5. Left rotate" << endl;
cout << "6. Quit" << endl;
```

The options should have the following behavior:

- **Find employee**: Prompts the user for `empName` and `empId`. Calls `searchEmployee` function with the provided arguments and calls `printEmployee` on the returned pointer.
- **Insert employee:** Prompts the user for necessary arguments. Calls `insertEmployee` function with the provided arguments.
- **Remove employee:** Prompts the user for `empName` and `empId.` Calls `removeEmployee` function with the provided arguments.

- **Show employee directory:** Call your tree's **`showEmployeeDirectory`** function
- **Left rotate:** Prompts the user for the **`lastInitial`** to locate the TreeNode to be rotated left. Calls **`searchCharNode`** to get a pointer to the node and calls **`leftRotate`** on it if it is a valid node (== not NULL).
- **Quit:** Exit after notifying the user:

```
cout << "Exiting program." << endl;
```

Please note that once you are done with your assignment on the autograder, you need to click on 'Finish Attempt' and then 'Submit all and finish'. If you don't do this, your attempt will not get graded.