CSCI 2270 – Data Structures

Instructors: Shayon Gupta, Ashutosh Trivedi, Maciej Zagrodzki

Assignment 6 - Binary Search Trees II

OBJECTIVES

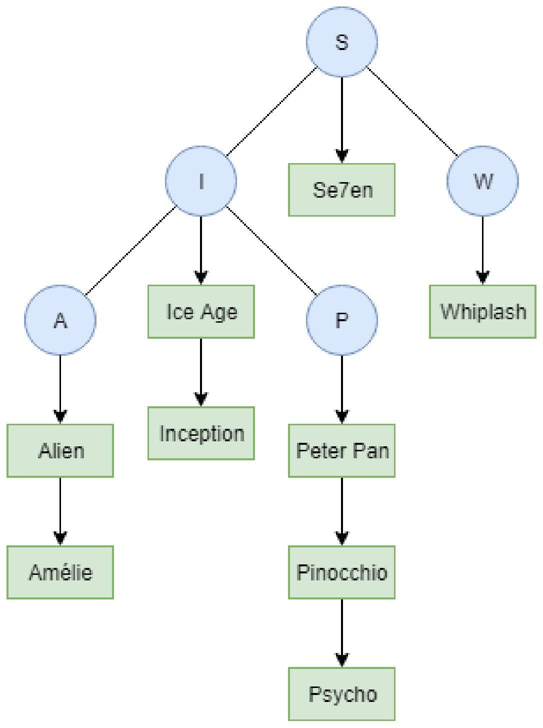
1. **Delete nodes in a BST**
2. **Create a super data structure combining BST and LL**

Overview

This assignment builds off of the previous one conceptually, although the data structure is fundamentally different. Make sure to use the ***MovieTree.hpp*** file uploaded on Moodle for assignment 6, **not** the ***MovieTree.hpp*** of the previous assignment 5. As usual, **do not** modify the header file. *You may implement helper functions in your .cpp file to facilitate recursion if you want as long as you don't add those functions to the MovieTree class*.

MovieTree Class

Your task is to implement a binary search tree of linked lists of movies. *Disclaimer: this diabolical super data structure is not practical; it is meant to challenge you and improve your skills manipulating and traversing BST's and LL's.* Tree nodes will contain a letter of the alphabet and a linked list. The linked list will be an alphabetically sorted list of movies which start with that letter. For example:



MovieTree()

➔ Constructor: Initialize any member variables of the class to default

~MovieTree()

➔ Destructor: Free all memory that was allocated

void printMovieInventory()

➔ Print every movie in the data structure in alphabetical order of titles using the following format. For TreeNode **t** and LLMovieNode **m**:

```
// for every TreeNode (t) in the tree
cout << "Movies starting with letter: " << t->titleChar << endl;
// for every LLMovieNode (m) attached to t
cout << " >> " << m->title << " " << m->rating << endl;
```

<u>**Sample output format**</u>

```
Movies starting with letter: B
 >> Bowling for Columbine 8
Movies starting with letter: D
 >> Dancin' Outlaw 8.1
 >> Dogtown and Z-Boys 7.7
 >> Down from the Mountain 7.4
```

void addMovie(int ranking, std::string title, int year, float rating)

➔ Add a movie to the data structure in the correct place based on its title.
- ◆ If there is no tree node corresponding to the first letter of title, create it and insert it in the tree in the alphabetically correct position
- ◆ Create a linked list node with ranking, title, year and rating, and insert it in the linked list associated with the tree node associated with the first letter of title. The linked list must also be in alphabetical order, such that for each **node,**

```
node->title < node->next->title
```

Hint: you can compare strings with <, >, ==, etc. Also, you may assume that no two movies have the same title

void deleteMovie(std::string title)

➔ Delete the linked list node that contains **title**. If as a result of this deletion, the linked list becomes empty, delete the associated tree node. If the movie does not exist in the data structure, print the following message

```
cout << "Movie: " << title << " not found, cannot delete." << endl;
```

Driver

Your main function should first read information about each movie from a file and store that information in a MovieTree object. **The name of the file with this information should be passed in as a command-line argument.** An example file is *Movies.csv* on Moodle. It is in the format:

```
<Movie 1 ranking>,<Movie 1 title>,<Movie 1 year>,<Movie 1 rating>
<Movie 2 ranking>,<Movie 2 title>,<Movie 2 year>,<Movie 2 rating>
Etc...
```

Note: For autograding's sake, insert the nodes to the tree in the order they are read in. **After** reading in the information on each movie from the file, display a menu to the user.

```
cout << "======Main Menu======" << endl;
cout << "1. Print the inventory" << endl;
cout << "2. Delete a movie" << endl;
cout << "3. Quit" << endl;
```

The options should have the following behavior:

- **Print the inventory:** Call your tree's **printMovieInventory** function
- **Delete a movie:** Call your **deleteMovie** function on a title specified by the user. Prompt the user for a movie title using the following code:

  ```
  cout << "Enter title:" << endl;
  ```

- **Quit:** Exit after printing a friendly message to the user:

  ```
  cout << "Goodbye!" << endl;
  ```