Ceng213 - Data Structures
Programming Assignment 2 : Book Catalog Implementation via Binary Search Trees

Spring 2021

1 Objectives

In this programming assignment, you are first expected to implement a *binary search tree* data structure, in which each node will contain an element and two pointers to the root nodes of its left and right subtrees. The binary search tree data structure will include a single pointer that points to the root node of the binary search tree. The details of the structure are explained further in the following sections. Then, you will use this specialized binary search tree structure to implement a *book catalog*.

Keywords: *C++, Data Structures, Binary Trees, Binary Search Trees, Book Catalog Implementation*

2 Binary Search Tree Implementation (60 pts)

The binary search tree data structure used in this assignment is implemented as the class template `BinarySearchTree` with the template argument `T`, which is used as the type of the data stored in the nodes. The node of the binary search tree is implemented as the class template `Node` with the template argument `T`, which is the type of the data stored in nodes. `Node` class is the basic building block of the `BinarySearchTree` class. `BinarySearchTree` class has a single `Node` pointer in its private data field (namely `root`) which points to the root node of the binary search tree.

The `BinarySearchTree` class has its definition and implementation in *BinarySearchTree.h* file and the `Node` class has its in *Node.h* file.

2.1 `Node`

`Node` class represents nodes that constitute binary search trees. A `Node` keeps two pointers (namely `left` and `right`) to the root nodes of its left and right subtrees, and a data variable of type `T` (namely `element`) to hold the data. The class has three constructors and the overloaded output operator. They are already implemented for you. You should not change anything in file *Node.h*.

2.2 BinarySearchTree

`BinarySearchTree` class implements a binary search tree data structure with the `root` pointer. Previously, data members of `BinarySearchTree` class have been briefly described. Their use will be elaborated in the context of utility functions discussed in the following subsections. You must provide implementations for the following public interface methods that have been declared under indicated portions of *BinarySearchTree.h* file.

2.2.1 `BinarySearchTree();`

This is the default constructor. You should make necessary initializations in this function.

2.2.2 `BinarySearchTree(const BinarySearchTree<T> &obj);`

This is the copy constructor. You should make necessary initializations, create new nodes by copying the nodes in given `obj`, and insert new nodes into the binary search tree. The structure among the nodes in given `obj` should also be copied to the binary search tree.

2.2.3 `~BinarySearchTree();`

This is the destructor. You should deallocate all the memory that you allocated before.

2.2.4 `Node<T> *getRoot() const;`

This function should return a pointer to the root node in the binary search tree. If the binary search tree is empty, it should return `NULL`.

2.2.5 `void insert(const T &element);`

You should create a new node with given `element` and insert it at the appropriate location in the binary search tree. Don't forget to make necessary pointer modifications in the tree. ***You will not be asked to insert duplicated elements to the binary search tree.***

2.2.6 `void remove(const T &element);`

You should remove the node with given `element` from the binary search tree. Don't forget to make necessary pointer modifications in the tree. If there exists no such node in the binary search tree, do nothing. ***There will be no duplicated elements in the binary search tree.***

2.2.7 `Node<T> *find(const T &element);`

You should search the binary search tree for the node that has the same element with the given `element` and return a pointer to that node. You can use the `operator==` to compare two `T` objects. If there exists no such node in the binary search tree, you should return `NULL`. ***There will be no duplicated elements in the binary search tree.***

2.2.8 `Node<T> *findWithoutOrderingProperty(const T &element);`

This function has almost same functionality with the previous `find` function. The only difference is that you should not take advantage of the ordering property of binary search trees to search for nodes (i.e., search both subtrees of nodes).

2.2.9 `std::vector<Node<T> *> getListOfAllNodes();`

You should return an `std::vector` of pointers to all nodes in the binary search tree. The nodes should be sorted in ascending order using `operator<` and `operator>` relational operators of their elements.

2.2.10 `void depthFirstTraversal() const;`

You should *preorder depth first traverse* all nodes of the binary search tree and print them to the standard output (`std::cout`). The format for printing in this function should be the same with the format used in the already implemented `operator<<` operator. *In this function, you should utilize `std::stack` to implement depth first traversal.*

2.2.11 `void breadthFirstTraversal() const;`

You should *breadth first traverse* all nodes of the binary search tree and print them to the standard output (`std::cout`). The format for printing in this function should be the same with the format used in the already implemented `operator<<` operator. *In this function, you should utilize `std::queue` to implement breadth first traversal.*

2.2.12 `BinarySearchTree<T> &operator=(const BinarySearchTree<T> &rhs);`

This is the overloaded assignment operator. You should remove all nodes in the binary search tree and then create new nodes by copying the nodes in given `rhs` and insert new nodes into the binary search tree. The structure among the nodes in given `rhs` should also be copied to the binary search tree.

3 Book Catalog Implementation (40 pts)

The book catalog in this assignment is implemented as the class `BookCatalog`. `BookCatalog` class has a `BinarySearchTree` object in its private data field (namely `categories`) with the type `Category`. This `BinarySearchTree` object keeps categories of the book catalog. `Category` class has a `BinarySearchTree` object in its private data field (namely `books`) with the type `Book`. These `BinarySearchTree` objects in `Category` objects keep books of the book catalog.

The `BookCatalog` class has its definition in *BookCatalog.h* file and its implementation in *BookCatalog.cpp* file. The `Category` class has its definition in *Category.h* file and its implementation in *Category.cpp* file. The `Book` class has its definition in *Book.h* file and its implementation in *Book.cpp* file.

3.1 `Book`

`Book` objects keep `isbn`, `title` and `author` variables of type `std::string` to hold the data related with the books in the book catalog. Most of the functions of `Book` class are already implemented for you. In *Book.cpp* file, you need to provide implementations for following functions declared under *Book.h* header to complete the assignment. You should not change anything in file *Book.h*.

In this assignment, we will have some assumptions for the books in book catalogs. First of all, no two books will have a common isbn or title (i.e., each book will have a unique isbn and title). They can only share a common author. Also, each book will belong to an only category. There

will be no books with multiple categories.

3.1.1 `bool operator<(const Book &rhs) const;`

This is the overloaded less than comparison operator. If `isbn` values of both this book and the given book (`rhs`) are not empty, you should compare them lexicographically. If either or both of `isbn` values are empty but both of `title` values are not empty, you should compare `title` values lexicographically. If either or both of `title` values are also empty but both of `author` values are not empty, you should compare `author` values lexicographically. If the compared value of this book is less than the given book's corresponding value, return `true`. Otherwise, return `false`.

3.1.2 `bool operator>(const Book &rhs) const;`

This is the overloaded greater than comparison operator. This function should follow the same rules to choose values to compare with `operator<` function, but uses `>` for comparison.

3.1.3 `bool operator==(const Book &rhs) const;`

This is the overloaded equality comparison operator. This function should follow the same rules to choose values to compare with `operator<` function, but uses `==` for comparison.

3.2 `Category`

`Category` objects keep `name` variable of type `std::string` to hold the data related with the categories in the book catalog. They also keep binary search trees of books with those categories (namely `books` variable of type `BinarySearchTree<Book>`). Most of the functions of `Category` class are already implemented for you. In *Category.cpp* file, you need to provide implementations for following functions declared under *Category.h* header to complete the assignment. You should not change anything in file *Category.h*.

3.2.1 `void addBook(const Book &book);`

This is the member function to insert a new book under this category of the book catalog. For this function, it is guaranteed that the given `book` object will include all of `isbn`, `title`, and `author` values. If there already exists a book with the same `isbn` value under this category, do nothing.

3.2.2 `void removeBook(const Book &book);`

This is the member function to remove an existing book from this category of the book catalog. If the given `book` object includes `isbn` value, you should search for the book to remove by isbn. If `isbn` value is not included but `title` value is included, you should search for the book to remove by title. For this function, it is guaranteed that the given `book` object will include either of `isbn` and `title` values. If there exists no such book under this category, do nothing.

3.2.3 `Book *searchBook(const Book &book);`

This is the member function to search for an existing book from this category of the book catalog and to return a pointer to that book. If the given `book` object includes `isbn` value, you should search for the book by isbn. If `isbn` value is not included but `title` value is included, you should

search for the book by title. For this function, it is guaranteed that the given `book` object will include either of `isbn` and `title` values. If there exists no such book under this category, you should return `NULL`.

3.2.4 `std::vector<Book *> getListOfBooks();`

You should return an `std::vector` of pointers to all books under this category of the book catalog. The books should be sorted in ascending order by their `isbn` values.

3.3 BookCatalog

In `BookCatalog` class, all member functions should utilize `categories` member variable to operate as described in the following subsections. In *BookCatalog.cpp* file, you need to provide implementations for following functions declared under *BookCatalog.h* header to complete the assignment.

3.3.1 `void addCategory(const std::string &categoryName);`

This function adds a new category to the book catalog. It takes the name of the category to add as parameter (`categoryName`) and inserts a new `Category` object to the `categories` binary search tree. If there already exists a category with the same name, do nothing.

3.3.2 `void removeCategory(const std::string &categoryName);`

This function removes an existing category from the book catalog. It takes the name of the category to remove as parameter (`categoryName`) and removes that `Category` object from the `categories` binary search tree. If there exists no such category, do nothing.

3.3.3 `Category *searchCategory(const std::string &categoryName);`

This function searchs for an existing category from the book catalog. It takes the name of the category to search as parameter (`categoryName`) and returns a pointer to that `Category` object from the `categories` binary search tree. If there exists no such category, you should return `NULL`.

3.3.4 `void addBook(const std::string &categoryName, const Book &book);`

This function adds a new book under a category of the book catalog. It takes the name of the category and the details of the book as parameters (`categoryName` and `book`) and inserts a new `Book` object to the `books` binary search tree of the `Category` object with name `categoryName`. If there exists no such category in the book catalog, you should add a new category with name `categoryName` at first. If there already exists a book with the same `isbn` under that category, do nothing.

3.3.5 `void removeBookByIsbn(const std::string &isbn);`

This function removes an existing book from a category of the book catalog. It takes the isbn of the book to remove as parameter (`isbn`) and removes that `Book` object from the `books` binary search tree of the `Category` object that includes the book. If there exists no such book, do nothing.

3.3.6 `void removeBookByTitle(const std::string &title);`

This function removes an existing book from a category of the book catalog. It takes the title of the book to remove as parameter (`title`) and removes that `Book` object from the `books` binary search tree of the `Category` object that includes the book. If there exists no such book, do nothing.

3.3.7 `Book *searchBookByIsbn(const std::string &isbn);`

This function searches for an existing book from a category of the book catalog. It takes the isbn of the book to search as parameter (`isbn`) and returns a pointer to that `Book` object from the `books` binary search tree of the `Category` object that includes the book. If there exists no such book, you should return `NULL`.

3.3.8 `Book *searchBookByTitle(const std::string &title);`

This function searches for an existing book from a category of the book catalog. It takes the title of the book to search as parameter (`title`) and returns a pointer to that `Book` object from the `books` binary search tree of the `Category` object that includes the book. If there exists no such book, you should return `NULL`.

3.3.9 `std::vector<Book *> getListOfBooksByCategory(const std::string &categoryName);`

You should return an `std::vector` of pointers to all books under a category with given name `categoryName` of the book catalog. The books should be sorted in ascending order by their `isbn` values.

3.3.10 `std::vector<Book *> getListOfBooksByAuthor(const std::string &author);`

You should return an `std::vector` of pointers to all books by given author `author` in the book catalog. The books should be sorted in ascending order by their `isbn` values.

4 Driver Programs

To enable you to test your `BinarySearchTree` and `BookCatalog` implementations, two driver programs, *main_binarysearchtree.cpp* and *main_bookcatalog.cpp* are provided. Their expected outputs are also provided in *output_binarysearchtree.txt* and *output_bookcatalog.txt* files, respectively.

5 Regulations

1. **Programming Language:** You will use C++.
2. **S**tandard Template Library is **not** allowed unless you are explicitly asked to use it for a task.
3. **E**xternal libraries other than those already included are **not** allowed.
4. **T**hose who do the operations (insert, remove, find, ...) without utilizing the linked list will receive **0 grade**.

5. **T**hose who modify already implemented functions and those who insert other data variables or public functions and those who change the prototype of given functions will receive **0 grade**.

6. **Y**ou can add private member functions whenever it is explicitly allowed.

7. **Late Submission Policy:** Each student receives 5 late days for the entire semester. You may use late days on programming assignments, and each allows you to submit up to 24 hours late without penalty. For example, if an assignment is due on Thursday at 11:30pm, you could use 2 late days to submit on Saturday by 11:30pm with no penalty. Once a student has used up all their late days, each successive day that an assignment is late will result in a loss of 5% on that assignment.

   No assignment may be submitted more than 3 days (72 hours) late without permission from the course instructor. In other words, this means there is a practical upper limit of 3 late days usable per assignment. If unusual circumstances truly beyond your control prevent you from submitting an assignment, you should discuss this with the course staff as soon as possible. If you contact us well in advance of the deadline, we may be able to show more flexibility in some cases.

8. **Cheating:** We have zero tolerance policy for cheating. In case of cheating, all parts involved (source(s) and receiver(s)) get zero. People involved in cheating will be punished according to the university regulations. Remember that students of this course are bounded to code of honor and its violation is subject to severe punishment.

9. **Newsgroup:** You must follow the Forum (`odtuclass.metu.edu.tr`) for discussions and possible updates on a daily basis.

6 Submission

- Submission will be done via CengClass (`cengclass.ceng.metu.edu.tr`).
- Don't write a main function in any of your source files.
- A test environment will be ready in CengClass.
  - You can submit your source files to CengClass and test your work with a subset of evaluation inputs and outputs.
  - Additional test cases will be used for evaluation of your final grade. So, your actual grades may be different than the ones you get in CengClass.
  - Only the last submission before the deadline will be graded.