DSA Lab: 11

Department of Computer Science

Iqra University Islamabad

DSA

Maqsood Ahmed

ID: 3818

Binary Search Tree:

Scenario: Library Book Management System

Imagine you are developing a Library Book Management System where each book is assigned a unique numeric Book ID. The system needs to efficiently manage the collection of books by supporting the following operations:

1. Insert: Add a new book to the library by its unique Book ID. If a book with the same ID already exists, the system should prevent duplicate entries.

2. Search: Allow library staff to search for a book by its Book ID to check its availability or fetch its details.

3. Update: Modify the details of a book (e.g., title, author, genre) if it exists in the library's collection.

4. Delete: Remove a book from the library's collection if it's no longer available or needed.

The library uses a Binary Search Tree (BST) to manage the books because it enables fast operations on the Book IDs, ensuring that the system remains efficient as the collection grows.

Implementation Logic:

1. Nodes:

Each node in the BST represents a book.

A node contains the following attributes:

- book_id: The unique numeric ID of the book (used as the key for BST operations).
- title: The title of the book.
- author: The author's name.
- genre: The genre of the book.
- left and right: Pointers to the left and right child nodes.

2. Insertion:

Add a new node for the book by traversing the BST.

Place the new book's node in the correct position based on its book_id.

3. Search:

Traverse the BST to locate a node with the given book_id.

If found, display the book details; otherwise, notify that the book is not available.

4. Update:

Search for the book by its book_id.

If the book exists, update its attributes like title, author, or genre.

5. Delete:

Locate the node with the given book_id.

Remove the node using the appropriate BST deletion rules:

- If the node is a leaf, delete it directly.
- If the node has one child, replace it with its child.
- If the node has two children, replace it with its in-order successor or predecessor.

Source Code:

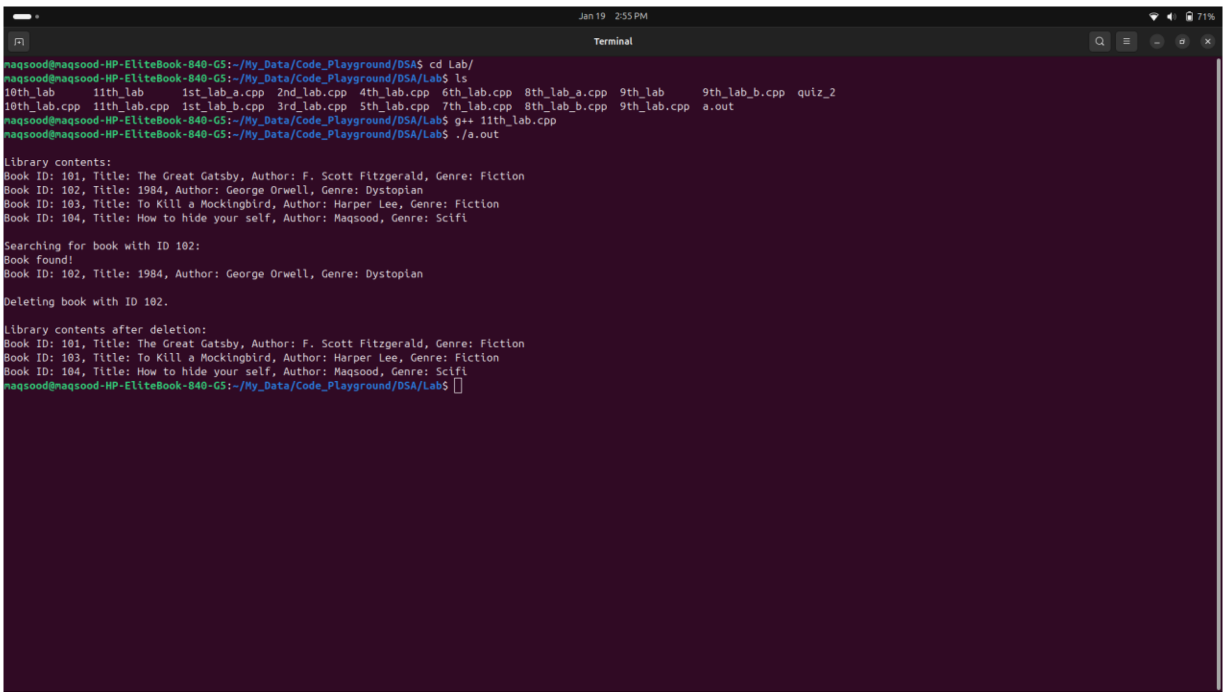
```
#include <iostream>
#include <string>
using namespace std;

class Node {
public:
    // book attributes
    int book_id;
    string title;
    string author;
    string genre;

    Node* left;
    Node* right;

    Node(int book_id, string title, string author, string genre) {
        this->book_id = book_id;
        this->title = title;
        this->author = author;
        this->genre = genre;
        this->left = nullptr;
        this->right = nullptr;
    }

    ~Node() {
        delete left;
        delete right;
    }
};

class BST {
private:
    Node* root;

    Node* insert(Node* node, int book_id, string title, string author, string genre) {
        if (node == nullptr) {
            return new Node(book_id, title, author, genre);
        }
        if (book_id < node->book_id) {
            node->left = insert(node->left, book_id, title, author, genre);
        } else if (book_id > node->book_id) {
            node->right = insert(node->right, book_id, title, author, genre);
        }       return node;
    }

    Node* findMin(Node* node) {
        while (node && node->left != nullptr) {
```

```
        node = node->left;
    }
    return node;
}

Node* deleteNode(Node* node, int book_id) {
    if (node == nullptr) return node;

    if (book_id < node->book_id) {
        node->left = deleteNode(node->left, book_id);
    } else if (book_id > node->book_id) {
        node->right = deleteNode(node->right, book_id);
    } else {
        // Node to be deleted found
        if (node->left == nullptr) {
            Node* temp = node->right;
            node->right = nullptr;
            delete node;
            return temp;
        } else if (node->right == nullptr) {
            Node* temp = node->left;
            node->left = nullptr;
            delete node;
            return temp;
        }
        // Node with two children
        Node* temp = findMin(node->right);
        node->book_id = temp->book_id;
        node->title = temp->title;
        node->author = temp->author;
        node->genre = temp->genre;
        node->right = deleteNode(node->right, temp->book_id);
    }
    return node;
}

Node* search(Node* node, int book_id) {
    if (node == nullptr || node->book_id == book_id) {
        return node;
    }
    if (book_id < node->book_id) {
        return search(node->left, book_id);
    }
    return search(node->right, book_id);
}

void display(Node* node) {
    if (node != nullptr) {
        display(node->left);
        cout << "Book ID: " << node->book_id
             << ", Title: " << node->title
```

```
            << ", Author: " << node->author
            << ", Genre: " << node->genre << endl;
        display(node->right);
    }
}

public:
BST() {
    root = nullptr;
}

~BST() {
    delete root;
}

void insert(int book_id, string title, string author, string genre) {
    root = insert(root, book_id, title, author, genre);
}

void deleteNode(int book_id) {
    root = deleteNode(root, book_id);
}

void search(int book_id) {
    Node* result = search(root, book_id);
    if (result != nullptr) {
        cout << "Book found!\n"
            << "Book ID: " << result->book_id
            << ", Title: " << result->title
            << ", Author: " << result->author
            << ", Genre: " << result->genre << endl;
    } else {
        cout << "Book with ID " << book_id << " not found." << endl;
    }
}

void display() {
    if (root == nullptr) {
        cout << "The library is empty." << endl;
    } else {
        display(root);
    }
}
};

int main() {
BST library;

library.insert(101, "The Great Gatsby", "F. Scott Fitzgerald", "Fiction");
library.insert(102, "1984", "George Orwell", "Dystopian");
library.insert(103, "To Kill a Mockingbird", "Harper Lee", "Fiction");
```

```
    library.insert(104, "How to hide your self", "Maqsood", "")

    cout << "\nLibrary contents:" << endl;
    library.display();

    cout << "\nSearching for book with ID 102:" << endl;
    library.search(102);

    cout << "\nDeleting book with ID 102." << endl;
    library.deleteNode(102);

    cout << "\nLibrary contents after deletion:" << endl;
    library.display();

    return EXIT_SUCCESS;
}
```

OUTPUT:



The End