**GROUP B**

**A book consists of chapters, chapters consist of sections and sections consist of subsections. Construct a tree and print the nodes. Find the time and space requirements of your method.**

#include <iostream>

#include <vector>

using namespace std;

class Node {

public:

string title;

vector<Node\*> children;

Node(const string& title)

: title(title) {}

~Node() {

for (Node\* child : children) {

delete child;

}

}

};

class Book {

private:

Node\* root;

public:

Book(const string& title)

: root(new Node(title)) {}

~Book() {

delete root;

}

void addChapter(const string& chapterTitle) {

Node\* chapter = new Node(chapterTitle);

root->children.push\_back(chapter);

}

void addSection(const string& chapterTitle, const string& sectionTitle) {

Node\* section = new Node(sectionTitle);

Node\* chapterNode = findChapter(chapterTitle);

if (chapterNode != nullptr) {

chapterNode->children.push\_back(section);

}

else {

cerr << "Chapter not found: " << chapterTitle << endl<<endl;

delete section;

}

}

void addSubsection(const string& chapterTitle, const string& sectionTitle, const string& subsectionTitle) {

Node\* subsection = new Node(subsectionTitle);

Node\* chapterNode = findChapter(chapterTitle);

if (chapterNode != nullptr) {

Node\* sectionNode = findSection(chapterNode, sectionTitle);

if (sectionNode != nullptr) {

sectionNode->children.push\_back(subsection);

}

else {

cerr << "\nSection not found: " << sectionTitle << " in Chapter: " << chapterTitle << endl<<endl;

delete subsection;

}

}

else {

cerr << "Chapter not found: " << chapterTitle << endl<<endl;

delete subsection;

}

}

Node\* findChapter(const string& chapterTitle) {

for (Node\* chapter : root->children) {

if (chapter->title == chapterTitle) {

return chapter;

}

}

return nullptr;

}

Node\* findSection(Node\* chapterNode, const string& sectionTitle) {

for (Node\* section : chapterNode->children) {

if (section->title == sectionTitle) {

return section;

}

}

return nullptr;

}

void printNodes(Node\* node) {

if (node == nullptr) {

return;

}

cout << node->title << endl;

for (Node\* child : node->children) {

printNodes(child);

}

}

void printBook() {

printNodes(root);

}

};

int main() {

Book book("My Book");

// Adding chapters, sections, and subsections

book.addChapter("\nChapter 1");

book.addSection("\nChapter 1", "Section 1.1");

book.addSubsection("\nChapter 1", "Section 1.1", "Subsection 1.1.1");

book.addSubsection("\nChapter 1", "Section 1.1", "Subsection 1.1.2");

book.addSection("\nChapter 1", "Section 1.2");

book.addChapter("\nChapter 2");

book.addSection("\nChapter 2", "Section 2.1");

// Attempt to add a subsection to a non-existent section

book.addSubsection("\nChapter 2", "Non-existent Section", "Subsection 2.1.1");

// Print the nodes of the book

book.printBook();

return 0;

}

/\*

OUTPUT:-

Section not found: Non-existent Section in Chapter:

Chapter 2

My Book

Chapter 1

Section 1.1

Subsection 1.1.1

Subsection 1.1.2

Section 1.2

Chapter 2

Section 2.1

/\*

Time Complexity:

Adding a chapter: O(1)

Adding a section: O(N)

Adding a subsection: O(N\*M)

Printing the book: O(N)

Space Complexity:

Tree structure: O(N)

Additional space for recursive function calls: Depends on the book structure and depth of recursion.