Lab task 10

#include <iostream>

#include <string>

Struct Node {

Std::string name;

Int semester;

Int sapId;

Node\* prev;

Node\* next;

};

Class DoublyLinkedList {

Public:

Node\* head;

DoublyLinkedList() {

Head = nullptr;

}

Void insertAtMiddle(const std::string& name, int semester, int sapId) {

Node\* newNode = new Node;

newNode->name = name;

newNode->semester = semester;

newNode->sapId = sapId;

newNode->prev = nullptr;

newNode->next = nullptr;

if (head == nullptr) {

head = newNode;

return;

}

Node\* slowPtr = head;

Node\* fastPtr = head;

While (fastPtr != nullptr && fastPtr->next != nullptr) {

slowPtr = slowPtr->next;

fastPtr = fastPtr->next->next;

}

// Insert the new node after slowPtr

newNode->next = slowPtr->next;

if (slowPtr->next != nullptr) {

slowPtr->next->prev = newNode;

}

newNode->prev = slowPtr;

slowPtr->next = newNode;

}

Void deleteByValue(const std::string& name) {

Node\* current = head;

While (current != nullptr) {

If (current->name == name) {

If (current == head) {

Head = current->next;

If (head != nullptr) {

Head->prev = nullptr;

}

} else {

Current->prev->next = current->next;

If (current->next != nullptr) {

Current->next->prev = current->prev;

}

}

Delete current;

Return;

}

Current = current->next;

}

Std::cout << “Value not found in the list.\n”;

}

Int countNodes() {

Int count = 0;

Node\* current = head;

While (current != nullptr) {

Count++;

Current = current->next;

}

Return count;

}

Void mergeLists(DoublyLinkedList& otherList) {

If (head == nullptr) {

Head = otherList.head;

Return;

}

Node\* tail = head;

While (tail->next != nullptr) {

Tail = tail->next;

}

Tail->next = otherList.head;

If (otherList.head != nullptr) {

otherList.head->prev = tail;

}

otherList.head = nullptr; // Clear the other list’s head

}

Void displayRecord() {

If (head == nullptr) {

Std::cout << “List is empty.\n”;

Return;

}

Node\* current = head;

While (current != nullptr) {

Std::cout << “Name: “ << current->name << std::endl;

Std::cout << “Semester: “ << current->semester << std::endl;

Std::cout << “SAP ID: “ << current->sapId << std::endl;

Std::cout << “--------------------\n”;

Current = current->next;

}

}

Void insertAtLocation(const std::string& name, int semester, int sapId, int position) {

If (position <= 0 || position > countNodes() + 1) {

Std::cout << “Invalid position.\n”;

Return;

}

If (position == 1) {

insertAtBeginning(name, semester, sapId);

return;

}

Node\* newNode = new Node;

newNode->name = name;

newNode->semester = semester;

newNode->sapId = sapId;

Node\* current = head;

Int count = 1;

While (count < position – 1) {

Current = current->next;

Count++;

}

newNode->next = current->next;

newNode->prev = current;

if (current->next != nullptr) {

current->next->prev = newNode;

}

Current->next = newNode;

}

Void insertAtBeginning(const std