LAB TASK 10

Name: Abdul Moiz

Sap: 54482

Semester: 3rd

Subject: DSA

Task:

```
#include <iostream>
#include <string>

using namespace std;

struct Student {
   string name;
   int semester;
   int sapID;
   Student* next;
   Student* prev;

   Student(string name, int semester, int sapID) : name(name),
semester(semester), sapID(sapID), next(nullptr), prev(nullptr) {}
};
```

```
class DoublyLinkedList {
private:
    Student* head;
    Student* tail;

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}

    void insertAtEnd(string name, int semester, int sapID) {
        Student* newNode = new Student(name, semester, sapID);
        if (!head) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }

    void insertAtMiddle(string name, int semester, int sapID) {
        Student* newNode = new Student(name, semester, sapID);
        if (!head) {
            head = tail = newNode;
```

```
        return;
    }

    int totalNodes = countNodes();
    int mid = totalNodes / 2;
    Student* temp = head;

    for (int i = 0; i < mid; i++) {
        temp = temp->next;
    }

    newNode->next = temp;
    newNode->prev = temp->prev;
    if (temp->prev) temp->prev->next = newNode;
    temp->prev = newNode;
    if (totalNodes % 2 == 0) {
        if (newNode->prev == nullptr) head = newNode;
    } else {
        if (temp == head) head = newNode;
    }
}

void deleteByValue(int sapID) {
    Student* temp = head;
```

```
    while (temp && temp->sapID != sapID) {
        temp = temp->next;
    }
    if (!temp) {
        cout << "Value not found." << endl;
        return;
    }

    if (temp->prev) temp->prev->next = temp->next;
    if (temp->next) temp->next->prev = temp->prev;
    if (temp == head) head = temp->next;
    if (temp == tail) tail = temp->prev;

    delete temp;
}

int countNodes() {
    int count = 0;
    Student* temp = head;
    while (temp) {
        count++;
        temp = temp->next;
    }
    return count;
```

```
    }

    void mergeLists(DoublyLinkedList& other) {
        if (!other.head) return;
        if (!head) {
            head = other.head;
            tail = other.tail;
        } else {
            tail->next = other.head;
            other.head->prev = tail;
            tail = other.tail;
        }
    }

    void display() {
        Student* temp = head;
        cout << "Student Records:\n";
        while (temp) {
            cout << "Name: " << temp->name << ", Semester: " << temp->semester <<
", SAP ID: " << temp->sapID << endl;
            temp = temp->next;
        }
    }

    void insertAtPosition(int position, string name, int semester, int sapID) {
```

```
if (position < 1) {
    cout << "Invalid position." << endl;
    return;
}

Student* newNode = new Student(name, semester, sapID);
if (position == 1) {
    newNode->next = head;
    if (head) head->prev = newNode;
    head = newNode;
    if (!tail) tail = newNode;
    return;
}

Student* temp = head;
for (int i = 1; i < position - 1 && temp; i++) {
    temp = temp->next;
}

if (!temp) {
    cout << "Position out of bounds." << endl;
    delete newNode;
    return;
}
```

```
    newNode->next = temp->next;
    if (temp->next) temp->next->prev = newNode;
    temp->next = newNode;
    newNode->prev = temp;

    if (newNode->next == nullptr) tail = newNode;
}

void deleteFromStart() {
    if (!head) return;
    Student* temp = head;
    head = head->next;
    if (head) head->prev = nullptr;
    else tail = nullptr;
    delete temp;
}

void deleteFromEnd() {
    if (!tail) return;
    Student* temp = tail;
    tail = tail->prev;
    if (tail) tail->next = nullptr;
    else head = nullptr;
```

```
        delete temp;
    }
};

int main() {
    DoublyLinkedList studentList;

    for (int i = 0; i < 7; i++) {
        string name;
        int semester, sapID;
        cout << "Enter student name, semester, and SAP ID: ";
        cin >> name >> semester >> sapID;
        studentList.insertAtEnd(name, semester, sapID);
    }

    int option;
    do {
        cout << "\nOptions:\n1. Insert at position\n2. Delete from start\n3. Delete from end\n4. Display records\n5. Insert at middle\n6. Delete by SAP ID\n7. Count nodes\n8. Exit\n";
        cin >> option;

        if (option == 1) {
            int pos, sem, sap;
            string name;
```

```
            cout << "Enter position, name, semester, and SAP ID: ";
            cin >> pos >> name >> sem >> sap;
            studentList.insertAtPosition(pos, name, sem, sap);
        } else if (option == 2) {
            studentList.deleteFromStart();
        } else if (option == 3) {
            studentList.deleteFromEnd();
        } else if (option == 4) {
            studentList.display();
        } else if (option == 5) {
            string name;
            int sem, sap;
            cout << "Enter name, semester, and SAP ID: ";
            cin >> name >> sem >> sap;
            studentList.insertAtMiddle(name, sem, sap);
        } else if (option == 6) {
            int sap;
            cout << "Enter SAP ID to delete: ";
            cin >> sap;
            studentList.deleteByValue(sap);
        } else if (option == 7) {
            cout << "Total nodes: " << studentList.countNodes() << endl;
        }
    } while (option != 8);
```

```
    return 0;
}
```