Áp dụng:

```
//element.h
#pragma once
#ifndef ELEMENT_H
#define ELEMENT_H


class element
{
private:
    int data;
    element* pointer;

public:
    element();
    element(int);
    virtual ~element();

    int Getdata() { return data; }
    void Setdata(int val) { data = val; }
    element* Getpointer() { return pointer; }
    void Setpointer(element* val) { pointer = val; }

protected:

};

#endif // ELEMENT_H



//element.cpp
#include "element.h"

element::element()
{
    //ctor
    this->data = 0;
    this->pointer = nullptr;
}
element::element(int data)
{
    //ctor
    this->data = data;
    this->pointer = nullptr;
}

element::~element()
{
    //dtor
}

//linkedlist.h
#ifndef LINKEDLIST_H
#define LINKEDLIST_H
#include "element.h"

class linkedlist
{
private:
    element* head;
```

```
    element* tail;
    int nNum;
public:
    linkedlist();
    virtual ~linkedlist();
    element* Gethead() { return head; }
    void Sethead(element* val) { head = val; }
    element* Gettail() { return tail; }
    void Settail(element* val) { tail = val; }
    void InsertFirst(element*);
    void InsertTail(element*);
    bool DeleteFirst();
    bool DeleteTail();
    int MaxList();
    void RemoveAll();
    int CountPrime();
    void InsertAfter(element* p, int value);
    void DeleteAfter(element* p);
    void Travel();
    int SumList();
    int CountValue(int x);
    bool Contains(int x);
    bool InsertUnique(int value);

    void SplitList(int x, linkedlist& smaller, linkedlist& larger);

protected:


};
#endif // LINKEDLIST_H

//linkedlist.cpp
#include "linkedlist.h"
#include <iostream>
using namespace std;
linkedlist::linkedlist()
{
    //ctor
    this->head = nullptr;
    this->tail = nullptr;
    this->nNum = 0;
}

linkedlist::~linkedlist()
{
    //dtor
}
void linkedlist::InsertFirst(element * e) {
    if (this->head == nullptr)
        this->head = this->tail = e;
    else {
        e->Setpointer(this->head); //step 1
        this->head = e;  // step 2
    }
    this->nNum++;

}
void linkedlist::InsertTail(element * e) {
    if (this->head == nullptr)
        this->head = this->tail = e;
    else {
        this->tail->Setpointer(e);// step 1
```

```
        this->tail = e;    // step 2
    }
    this->nNum++;
}
void linkedlist::Travel() {
    element* p = this->head;
    while (p != nullptr) {
        cout << p->Getdata() << "\t";
        p = p->Getpointer();
    }
}
int linkedlist::SumList()
{
    int sum = 0;
    element* p = this->head;
    while (p != nullptr) {
        sum += p->Getdata();
        p = p->Getpointer();
    }
    return sum;
}
bool linkedlist::DeleteFirst() {
    if (this->head == nullptr) return false;
    else {
        element* p = this->head;
        this->head = this->head->Getpointer();
        delete p;
        return true;
    }
}

bool linkedlist::DeleteTail() {
    if (this->head == nullptr) return false;
    else if (this->head == this->tail) {
        DeleteFirst();
    }
    else {
        element* p = this->head;
        while (p->Getpointer() != this->tail) {
            p = p->Getpointer();
        }
        delete this->tail;
        this->tail = p;
        this->tail->Setpointer(nullptr);
    }
    this -> nNum--;
    return true;
}

int linkedlist::MaxList() {
    if (this->head == nullptr) {
        cout << "Error.";
    }
    int max_ = this->head->Getdata();
    element* p = this->head->Getpointer();
    while (p != nullptr) {
        if (p->Getdata() > max_) {
            max_ = p->Getdata();
        }
        p = p->Getpointer();
    }
    return max_;
}
```

```
void linkedlist::RemoveAll() {
    element* current = this->head;
    while (current != nullptr) {
        element* next = current->Getpointer();
        delete current;
        current = next;
    }
    this->head = nullptr;
    this->tail = nullptr;
    this->nNum = 0;
}

bool isPrime(int num) {
        if (num < 2)
            return false;

        for (int i = 2; i <= sqrt((float)num); i++)
        {
            if (num % i == 0)
            {
                return false;
            }
        }
        return true;
}

int linkedlist::CountPrime() {
    int count = 0;
    element* p = this->head;
    while (p != nullptr) {
        if (isPrime(p->Getdata())) {
            count++;
        }
        p = p->Getpointer();
    }
    return count;
}

void linkedlist::InsertAfter(element* p, int value) {
    if (p == nullptr) {
        cout<<"Error.";
    }

    element* e = new element(value);
    e->Setpointer(p->Getpointer());

    p->Setpointer(e);

    if (p == tail) {
        tail = e;
    }

    this->nNum++;
}

bool linkedlist::DeleteAfter(element* p) {
    if (p == nullptr || p->Getpointer() == nullptr) {
        return false;
    }

    element* temp = p->Getpointer();
    p->Setpointer(temp->Getpointer());
```

```
    if (temp == this->tail) {
        this->tail = p;
    }

    delete temp;
    this->nNum--;
    return true;
}

int linkedlist::CountValue(int x) {
    int count = 0;
    element* p = this->head;
    while (p != nullptr) {
        if (p->Getdata() == x) {
            count++;
        }
        p = p->Getpointer();
    }
    return count;
}

bool linkedlist::Contains(int x) {
    element* p = this->head;
    while (p != nullptr) {
        if (p->Getdata() == x) {
            return true;
        }
        p = p->Getpointer();
    }
    return false;
}

bool linkedlist::InsertUnique(int value) {
    if (Contains(value)) {
        return false;
    }
    InsertTail(value);
    return true;
}

void linkedlist::SplitList(int x, linkedlist& smaller, linkedlist& larger) {
    element* p = this->head;
    while (p != nullptr) {
        if (p->Getdata() < x) {
            smaller.InsertTail(p->Getdata());
        }
        else {
            larger.InsertTail(p->Getdata());
        }
        p = p->Getpointer();
    }
}
```

Bài tập ứng dụng:

```
//Book.h
#pragma once
#include <string>
#include <vector>
#include <iostream>

using namespace std;

class Book
{
private:
	string title;
	vector<string> authors;
	string publisher;
	int year;
	Book* next;
public:
	Book(string title, vector<string> authors, string publisher, int year)
		:title(title), authors(authors), publisher(publisher), year(year),
next(nullptr) {}
	string GetTitle() const { return title; }
	vector<string> GetAuthors() const { return authors; }
	string GetPublisher() const { return publisher; }
	int GetYear() { return year; }
	Book* GetNext() const { return next; }
	void SetNext(Book* next) { this->next = next; }
};
```

```
//linkedlist.h
#pragma once
#include "Book.h"
#include <string>
#include <vector>
#include <iostream>

using namespace std;

class linkedlist {
private:
    Book* head;
    Book* tail;
    int nNum;
public:
    linkedlist();
    ~linkedlist();

    void InsertFirst(Book* book);
    void InsertTail(Book* book);
    void Travel() const;
    int CountBooksByAuthor(const string& author) const;
    vector<string> FindBooksByPublisherAndYear(const string& publisher, int year)
const;
};
```

```
//linkedlist.cpp
#include "linkedlist.h"
#include <iostream>
using namespace std;
linkedlist::linkedlist()
```

```
{
    //ctor
    this->head = nullptr;
    this->tail = nullptr;
    this->nNum = 0;
}

linkedlist::~linkedlist()
{
    //dtor
}
void linkedlist::InsertFirst(Book* book) {
    if (this->head == nullptr)
        this->head = this->tail = book;
    else {
        book->SetNext(this->head);
        this->head = book;
    }
    this->nNum++;
}

void linkedlist::InsertTail(Book* book) {
    if (this->head == nullptr)
        this->head = this->tail = book;
    else {
        this->tail->SetNext(book);
        this->tail = book;
    }
    this->nNum++;
}

void linkedlist::Travel() const {
    Book* p = this->head;
    while (p != nullptr) {
        cout << "Title: " << p->GetTitle() << ", Publisher: " << p->GetPublisher()
<< ", Year: " << p->GetYear() << "\n";
        p = p->GetNext();
    }
}

int linkedlist::CountBooksByAuthor(const string& author) const {
    int count = 0;
    Book* p = this->head;
    while (p != nullptr) {
        for (const auto& a : p->GetAuthors()) {
            if (a == author) {
                count++;
                break;
            }
        }
        p = p->GetNext();
    }
    return count;
}

vector<string> linkedlist::FindBooksByPublisherAndYear(const string& publisher,
int year) const {
    vector<string> books;
    Book* p = this->head;
    while (p != nullptr) {
        if (p->GetPublisher() == publisher && p->GetYear() == year) {
            books.push_back(p->GetTitle());
        }
```

```
        p = p->GetNext();
    }
    return books;
}

//main.cpp
#include "linkedlist.h"
#include <iostream>
using namespace std;

int main() {
    linkedlist library;
    int n;

    cout << "Enter the number of books: ";
    cin >> n;
    cin.ignore();

    for (int i = 0; i < n; ++i) {
        string title, publisher;
        int year, authorCount = 5;
        vector<string> authors;

        cout << "Enter the title of book " << i + 1 << ": ";
        getline(cin, title);

        for (int j = 0; j < authorCount; ++j) {
            string author;
            cout << "Enter author " << j + 1 << ": ";
            getline(cin, author);
            authors.push_back(author);
        }

        cout << "Enter the publisher of the book: ";
        getline(cin, publisher);

        cout << "Enter the year of publication: ";
        cin >> year;
        cin.ignore();

        library.InsertTail(new Book(title, authors, publisher, year));
    }

    string authorToFind;
    cout << "Enter the author to find the number of books: ";
    getline(cin, authorToFind);
    int authorCount = library.CountBooksByAuthor(authorToFind);
    cout << "Number of books by " << authorToFind << ": " << authorCount << endl;

    string publisherToFind;
    int yearToFind;
    cout << "Enter the publisher to find books: ";
    getline(cin, publisherToFind);
    cout << "Enter the year of publication: ";
    cin >> yearToFind;

    vector<string> booksByPublisherAndYear =
library.FindBooksByPublisherAndYear(publisherToFind, yearToFind);
    cout << "Books published by " << publisherToFind << " in " << yearToFind <<
":" << endl;
    for (const auto& book : booksByPublisherAndYear) {
        cout << book << endl;
    }
```

```
    return 0;
}
```