Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту



**Звіт**

про виконання

**Лабораторних та практичних робіт № 6**

***з дисципліни:*** «Мови та парадигми програмування»

***з розділу***: «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур»

***Виконала:***

студент групи ШІ-13

Жмуд Анастасія

# **Тема роботи:**

Інформаційні динамічні структури. Бінарні дерева.

# **Мета роботи:**

Знайомство з динамічними інформаційними структурами на прикладі одно- і двонаправлених списків та бінарних дерев. Дослідження їх властивостей, оптимізації та застосування для ефективного зберігання та обробки динамічної інформації.

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Однонаправлений списк
* Тема №2: Двонаправлений список
* Тема №3: Бінарні дерева
* Тема №4: Структури

1. Індивідуальний план опрацювання теорії:

* Тема №1: Однонаправлений списк
  + Джерела Інформації
    - https://www.geeksforgeeks.org/data-structures/linked-list/
    - https://www.programiz.com/dsa/linked-list
    - https://www.geeksforgeeks.org/what-is-linked-list/
  + Що опрацьовано:
    - Створення однонаправленого списку
    - Доступ до елементів однонаправленого списку та їх перебір
  + Статус: Ознайомлений
  + Початок опрацювання теми: 04.12.23
  + Звершення опрацювання теми: 06.12.23
* Тема №2: Двонаправлений список
  + Джерела Інформації:
    - https://www.geeksforgeeks.org/introduction-and-insertion-in-a-doubly-linked-list/
    - https://www.programiz.com/dsa/doubly-linked-list
    - https://www.tutorialspoint.com/cplusplus-program-to-implement-doubly-linked-list
  + Що опрацьовано:
    - Поняття двонаправленого списку
    - Створення вузлів та доступ до елементів
  + Статус: Ознайомлений
  + Початок опрацювання теми: 07.12.23
  + Звершення опрацювання теми: 08.12.23
* Тема №3: Бінарні дерева
  + Джерела Інформації:
    - https://purecodecpp.com/uk/archives/2483
    - <https://www.programiz.com/dsa/binary-tree>
    - https://www.geeksforgeeks.org/binary-tree-data-structure/
  + Що опрацьовано:
    - Елементи бінарного дерева, його формування
    - Виконання операцій з бінарними деревами
  + Статус: Ознайомлений
  + Початок опрацювання теми: 10.12.23
  + Звершення опрацювання теми: 11.12.23
* Тема №4: Структури
  + Джерела Інформації:
    - <https://acode.com.ua/urok-64-struktury/>
    - http://cpp.dp.ua/vykorystannya-struktur/
  + Що опрацьовано:
    - Ініціалізація структур
    - Доступ до членів структур
  + Статус: Ознайомлений
  + Початок опрацювання теми: 09.12.23
  + Звершення опрацювання теми: 10.12.23

# **Виконання роботи:**

## **1. Опрацювання завдання та вимог до програм та середовища:**

**Завдання №1 Опрацювання теорії**

* Деталі завдання:

Опрацювання різноманітних матеріалів, аналіз та дослідження відео, статей, книг на задані теми. Організація та структурування отриманих даних для можливості ефективніше засвоювати отримані знання на практиці. Вивчення найважливіших моментів.

**Завдання №2 Перегляд вимог та проектування**

* Деталі завдання

Введення завдань в Trello, перегляд задач, та створення математичної моделі. Проектування блок-схем в Draw.io.

**Завдання №3 VNS Lab 10**

* Варіант завдання: 23
* Деталі завдання:

Написати програму для запису в лінійному списку містять ключове поле типу \*char (рядок символів). Сформувати двонаправлений список. Знищити елемент із заданим

ключем. Додати К елементів після елемента із заданим ключем.

* Важливі деталі для врахування в імплементації програми: забезпечення коректної роботи двонаправленого списку

**Завдання №4 Algotester Lab 5**

* Варіант завдання: 2
* Деталі завдання:

В пустелі існує незвичайна печера, яка є двохвимірною. Її висота це

N, ширина – M. Всередині печери є пустота, пісок та каміння. Пустота позначається буквою О, пісок S і каміння X.

Одного дня стався землетрус і весь пісок посипався вниз. Він падає на найнижчу клітинку з пустотою, але він не може пролетіти через каміння.

Ваше завдання сказати як буде виглядати печера після землетрусу.

**Завдання № 5 Algotester Lab 7-8**

* Варіант завдання: 1
* Деталі завдання:

Ваше завдання - власноруч реалізувати структуру даних "Двозв’язний список".

Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його аргументи.

**Завдання №6 Class Practice Task**

* Деталі завдання:

Забезпечити роботу 5 програм: «Реверс списку», «Порівняння списків», «Додавання великих чисел», «Віддзеркалення дерева», «Записати кожному батьківському вузлу суму підвузлів»

**Завдання №7 Self Practice Task**

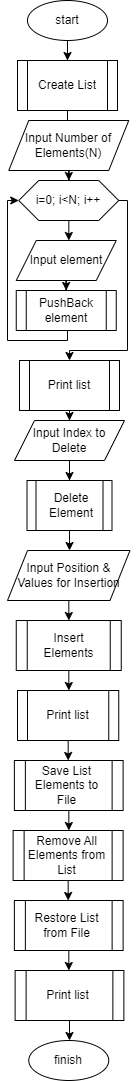
* Деталі завдання:

Виконати перевірку чи дерево є збалансованим чи ні.

## **2. Дизайн та планована оцінка часу виконання завдань:**

**Програма №1 VNS Lab 10**

* Блок-схема:



* Планований час на реалізацію: 5 годин
* Важливі деталі для врахування в імплементації:

Врахування того, що список необхідно створювати елементами структури

**Програма №2 Algotester Lab 5**

* Планований час на реалізацію: 3 години
* Важливі деталі для врахування в імплементації: необхідно кожен раз після переміщення піску перевіряти, чи випадково не залишилось піску зверху, щоб точно перемістити усе

**Програма №3 Algotester Lab 7-8**

* Планований час на реалізацію: 10 годин
* Важливі деталі для врахування в імплементації: Структура повинна бути написана як шаблон класу

**Програма №4 Class Practice Task**

* Планований час на реалізацію: 4 години
* Важливі деталі для врахування в імплементації:

Робота зі списками та деревами вимагає створення структури, що містить в собі показники на попередній і наступний елемент, якщо це двонаправлений список, показник на наступний елемент, якщо це однонапрямлений список і показники на підвузли.

**Програма №5 Self Practice Task**

* Планований час на реалізацію: 2 години
* Важливі деталі для врахування в імплементації:

Балансоване дерево: для кожного вузла різниця у висоті його лівого та правого піддерева не перевищує 1.

## **3. Код програм з посиланням на зовнішні ресурси:**

**Завдання №1 VNS Lab 10**

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

struct Node {

    string element;

    Node\* nx;

    Node\* pr;

};

struct LinkList {

    size\_t size;

    Node\* head;

    Node\* tail;

};

LinkList\* create() {

    LinkList\* m = new LinkList;

    m->size = 0;

    m->head = m->tail = nullptr;

    return m;

}

void print(LinkList\* list) {

    Node\* p = list->head;

    if (p == nullptr)

        cout << "Empty!";

    else {

        while (p != nullptr) {

            cout << p->element << " ";

            p = p->nx;

        }

    }

}

void pushBack(LinkList\* list, const string& value) {

    Node\* m = new Node;

    if (m == nullptr) {

        exit(3);

    }

    m->element = value;

    m->nx = nullptr;

    m->pr = list->tail;

    if (list->tail != nullptr) {

        list->tail->nx = m;

    }

    list->tail = m;

    if (list->head == nullptr) {

        list->head = m;

    }

    list->size++;

}

void saveToFile(LinkList\* list, const char\* file) {

    ofstream File(file);

    if (!File.is\_open()) {

        cout << "Error opening file!" << endl;

        return;

    }

    Node\* m = list->head;

    while (m != nullptr) {

        File << m->element << endl;

        m = m->nx;

    }

    File.close();

}

void restoration(const char\* file, LinkList\* list) {

    ifstream File(file);

    if (!File.is\_open()) {

        cout << "Error opening file!" << endl;

        return;

    }

    string s;

    while (getline(File, s)) {

        pushBack(list, s);

    }

    File.close();

}

void delet(LinkList\*\* list) {

    Node\* m = (\*list)->head;

    Node\* nex = nullptr;

    while (m) {

        nex = m->nx;

        delete m;

        m = nex;

    }

    delete\* list;

    (\*list) = nullptr;

}

string popBack(LinkList\* list) {

    if (list->tail == nullptr) {

        exit(4);

    }

    string element = list->tail->element;

    Node\* toDelete = list->tail;

    if (list->tail == list->head) {

        list->head = nullptr;

        list->tail = nullptr;

    } else {

        list->tail = list->tail->pr;

        if (list->tail != nullptr) {

            list->tail->nx = nullptr;

        }

    }

    delete toDelete;

    list->size--;

    return element;

}

Node\* getElement(LinkList \*list, size\_t index) {

    Node \*m = list->head;

    size\_t i = 0;

    while (m && i < index) {

    m =m->nx;

    i++;

    }

    return m;

}

string deleteElement(LinkList \*list, size\_t index) {

    list->size=list->size + 2;

    Node \*elm = NULL;

    elm = getElement(list, index);

    if (elm == NULL) {

    exit(5);

    }

    if (elm->pr) {

    elm->pr->nx = elm->nx;

    }

    if (elm->nx) {

    elm->nx->pr = elm->pr;

    }

    string m = elm->element;

    if (!elm->pr) {

    list->head = elm->nx;

    }

    if (!elm->nx) {

    list->tail = elm->pr;

    }

    free(elm);

    list->size--;

    return m;

}

   void insert(LinkList \*list, size\_t index, string value) {

    Node \*elm = NULL;

    Node \*ins = NULL;

    elm = getElement(list, index);

    if (elm == NULL) {

        exit(5);

    }

    ins = new Node;

    ins->element = value;

    ins->pr = elm;

    ins->nx = elm->nx;

    if (elm->nx) {

        elm->nx->pr = ins;

    }

    elm->nx = ins;

    if (!elm->pr) {

        list->head = elm;

    }

    if (!elm->nx) {

        list->tail = elm;

    }

    list->size++;

}

int main() {

    int n, f=0;

    size\_t index, num;

    string value1, value2;

    LinkList\* list = create();

    cout << "How many elements do you want to add?   ";

    cin >> n;

    for (int i = 0; i < n; i++){

        string k;

        cin >> k;

        pushBack(list, k);

        f++;

    }

    cout<<"List:  "; print(list); cout<<endl;

    cout<<"Enter index of element you want to delete: ";

    cin>>index;

    deleteElement(list, index-1);

    cout<<"List after change:  ";

    print(list);

    cout<<endl<<"Enter a number of element before adn after you want to add new element:  ";

    cin>>num;

    cout<<endl<<"Enter a first value you want to add:  ";

    cin>>value1;

    insert(list, num-2, value1);

    cout<<"Enter a second value you want to add:  ";

    cin>>value2;

    insert(list, num, value2);

    print(list); cout<<endl;

    const char\* file = "file.txt";

    saveToFile(list, file);

    for(int i=0; i<f; i++){

        popBack(list);

    }

    popBack(list);

    print(list);

    restoration(file, list);

    cout<<endl<<"File after restoration:  "; print(list);

    delet(&list);

    return 0;

}

Перш за все оголошуємо бібліотеки для взаємодії зі стандартним вводом/виводом (iostream), рядками (string), та роботи з файлами (fstream). Далі Створюються дві структури: Node та LinkList. Node представляє вузол списку з полями для елементу, посилань на наступний і попередній вузли. LinkList містить інформацію про розмір списку та посилання на початковий і кінцевий вузли.

Після, створюємо функцію create(), яка виділяє пам'ять під порожній список, ініціалізує розмір як 0 та встановлює посилання на початковий і кінцевий вузли як nullptr. Наступним етапом реалізується функція print(), що виводить елементи списку на екран. В ній використовується цикл для перегляду кожного вузла списку та виводу його значення.

Функція pushBack(), яка додає новий вузол із значенням value в кінець списку. Якщо список порожній, вузол стає і початковим, і кінцевим. Якщо ні, вузол додається до кінця списку. Після, реалізована функція saveToFile(), яка записує елементи списку у текстовий файл. Цикл проходиться по списку та записує кожне значення у файл у окремому рядку. Функція restoration() відкриває файл, з якого зчитується кожен рядок і додається як новий вузол у список. Далі, створюється функція deleteElement(), що видаляє елемент зі списку за вказаним індексом. Здійснюється пошук вузла за індексом, видалення посилань на цей вузол та звільнення пам'яті. Після, реалізується функція insert(), яка додає новий вузол з певним значенням перед або після вузла з вказаним індексом.

В функції main, Користувач вводить кількість елементів для додавання до списку, значення кожного з них. Відбувається видалення елемента за вказаним індексом та додавання нових елементів перед і після певного елемента. Список зберігається у файл file.txt. Далі відбувається процес відновлення списку із файлу. Останнім етапом є оголошення функції delet(), що видаляє всі вузли списку та звільняє пам'ять, що була використана для них.

**Завдання №2 Algotester Lab 5**

#include <iostream>

using namespace std;

int main() {

    int N, M;

    cin >> N >> M;

    char \*\*array = new char\*[N];

    for (int i = 0; i < N; ++i) {

        array[i] = new char[M];

    }

    for (int i = 0; i < N; ++i) {

        for (int j = 0; j < M; ++j) {

            cin >> array[i][j];

        }

    }

    for (int j = 0; j < M; ++j) {

        int k = N - 1;

        for (int i = N - 1; i >= 0; --i) {

            if (array[i][j] == 'S') {

                array[i][j] = 'O';

                if (k >= 0) {

                    array[k][j] = 'S';

                    k--;

                }

            } else if (array[i][j] == 'X') {

                k = i - 1;

            }

        }

    }

    for (int i = 0; i < N; ++i) {

        for (int j = 0; j < M; ++j) {

            cout << array[i][j];

        }

        cout << endl;

    }

    for (int i = 0; i < N; ++i) {

        delete[] array[i];

    }

    delete[] array;

    return 0;

}

Спочатку під’єднуємо усі необхідні бібліотеки, у нашому випадку це тільки “iostream”. Після, Користувач вводить значення N і M, що представляють розміри двовимірного масиву. Динамічно виділяється пам'ять для двовимірного масиву array розміром N на M. Д

Далі, відбувається ввід кожного елемента масиву за допомогою вкладених циклів for. Наступною нашою дією буде ітерація по кожному стовпцю масиву. Якщо зустрічається символ 'S', то він замінюється на 'O', а попередній 'S' (якщо він є) зміщується вище за поточний 'S' у стовпці, якщо є вільне місце для зсуву (тобто вище не знаходиться символ 'X'). Якщо зустрічається символ 'X', знаходимо останнє його входження у стовпці і зміщуємо "нижче" початок рядку від цього місця. Виводиться змінений двовимірний масив символів у консоль.

І на останок необхідно звільнити пам’ять, яка була виділена під кожен рядок, а потім під сам двовимірний масив array.

**Завдання №2 Algotester Lab 7-8**

#include <iostream>

#include <string>

using namespace std;

struct Node {

    int element;

    Node\* next;

    Node\* prev;

};

class List {

    public:

    int size;

    Node\* head;

    Node\* tail;

    List() {

        head = NULL;

        tail = NULL;

        size = 0;

    }

    void clear() {

        Node \*temp;

        while (head != NULL) {

            temp = head;

            head = head->next;

            delete temp;

        }

        size=0;

    }

    Node\* getNode(int index) {

        if (index < 0 || index >= size)

            return NULL;

        Node \*m = head;

        for (int i = 0; i < index; ++i) {

            m = m->next;

        }

        return m;

    }

    void insert(int index, int value) {

        if (index < 0 || index > size)

            return;

        Node \*elm = getNode(index - 1);

        Node \*ins = new Node;

        ins->element = value;

        ins->prev = NULL;

        ins->next = NULL;

        if (elm == NULL) {

            ins->next = head;

            if (head != NULL) {

                head->prev = ins;

            }

            head = ins;

            if (tail == NULL) {

                tail = ins;

            }

        } else {

            ins->next = elm->next;

            ins->prev = elm;

            if (elm->next != NULL) {

                elm->next->prev = ins;

            } else {

                tail = ins;

            }

            elm->next = ins;

        }

        size++;

    }

    void erase(int index) {

        if (index < 0 || index >= size)

            return;

        Node \*elm = getNode(index);

        if (elm == NULL) {

            return;

        }

        if (elm->prev != NULL) {

            elm->prev->next = elm->next;

        } else {

            head = elm->next;

            if (head != NULL) {

                head->prev = NULL;

            }

        }

        if (elm->next != NULL) {

            elm->next->prev = elm->prev;

        } else {

            tail = elm->prev;

            if (tail != NULL) {

                tail->next = NULL;

            }

        }

        delete elm;

        size--;

    }

    void print(ostream &s) {

        Node \*m = head;

        while (m != NULL) {

            s << m->element << ' ';

            m = m->next;

        }

    }

    int Size() {

        return size;

    }

};

int main() {

    List list;

    int q;

    cin >> q;

    for (int i = 0; i < q; i++) {

        string command;

        cin >> command;

        switch (command[0]) {

            case 'i': {

                int index, N;

                cin >> index >> N;

                for (int j = 0; j < N; j++) {

                    int element;

                    cin >> element;

                    list.insert(index + j, element);

                }

                break;

            }

            case 'p': {

                list.print(cout);

                cout << endl;

                break;

            }

            case 'e': {

                int in, n;

                cin >> in >> n;

                for (int j = 0; j < n; j++) {

                    list.erase(in);

                }

                break;

            }

            case 's': {

                if (command[1] == 'e') {

                    int ind, value;

                    cin >> ind >> value;

                    list.erase(ind);

                    list.insert(ind, value);

                } else {

                    cout << list.Size() << endl;

                }

                break;

            }

            case 'g': {

                int Index;

                cin >> Index;

                Node\* m = list.getNode(Index);

                if (m != NULL) {

                    cout << m->element << endl;

                } else {

                    cout << "Element not found" << endl;

                }

                break;

            }

        }

    }

    list.clear();

    return 0;

}

Для початку під’єднуємо бібліотеки «iostream» та «string». Далі оголошена структура, що містить в собі дані та посилання на попередній та наступний елементи. В класі List відбуваються наступні дії: Створюється двонаправлений список, кожен вузол списку містить ціле число (int element) і посилання на попередній і наступний вузли (вказівники Node\* next та Node\* prev). Клас List також містить методи для додавання, видалення, отримання розміру списку (insert, erase, size), виведення на екран (print) та очищення списку (clear). Головна функція main читає команди з консолі та виконує відповідні операції на зв'язаному списку. Команди можуть бути insert, erase, print, size, get та set, кожна з яких має свої власні визначені дії відповідно до специфікації.

**Завдання №4 Class Practice Task 1**

#include <iostream>

using namespace std;

struct Node {

    int data;

    Node\* next;

    Node(int value) : data(value),next(nullptr) {}

};

void add(Node\* &head, int value){

    Node\* m = new Node(value);

    m->next = head;

    head = m;

}

void print(Node\* m) {

    while (m != nullptr) {

        cout << m->data << " ";

        m = m->next;

    }

    cout << endl;

}

Node\* reverse(Node\* head) {

    Node\* prev = nullptr;

    Node\* current = head;

    Node\* next = nullptr;

    while (current != nullptr) {

        next = current->next;

        current->next = prev;

        prev = current;

        current = next;

    }

    return prev;

}

int main() {

    int N;

    Node\* head = nullptr;

    cout<<"How many numbers you want to add?  ";

    cin>>N;

    cout<<"Enter a numbers:  ";

    for(int i=0; i<N; i++) {

        int k;

        cin>>k;

        add(head, k);}

    head = reverse(head);

    cout<<"Initial list:  ";

    print(head);

    head = reverse(head);

    cout << "Revered list:  ";

    print(head);

}

Спочатку оголошуються бібліотека «iostream». Далі Структура Node містить два поля: int data, що зберігає ціле число, та вказівник Node\* next, що вказує на наступний вузол у списку. Крім того, визначений конструктор, який дозволяє ініціалізувати поле data та встановлює початкове значення вказівника next як nullptr. Функція add додає новий вузол із заданим значенням value на початок списку за допомогою вказівника head. Функція print виводить на екран значення усіх вузлів у зв'язаному списку, починаючи з вказаного вузла m та переходячи до наступних вузлів, поки не досягне кінця списку (вказівник next дорівнює nullptr). Функція reverse перевертає зв'язаний список. Вона працює шляхом зміни посилань на наступні та попередні вузли у списку таким чином, що кінцевий вузол стає початковим.

У функції main спочатку вводиться кількість чисел N, а потім вводяться ці числа для створення початкового списку. Створений список обертається за допомогою функції reverse та виводиться на екран як початковий список. Після цього список знову перевертається за допомогою reverse і виводиться як перевернутий список.

**Завдання №5 Class Practice Task 2**

#include <iostream>

using namespace std;

struct Node {

    int data;

    Node\* next;

    Node(int value) : data(value),next(nullptr) {}

};

void add(Node\* &head, int value){

    Node\* m = new Node(value);

    m->next = head;

    head = m;}

bool compare(Node\* h1, Node\* h2) {

    while (h1 != nullptr && h2 != nullptr) {

        if (h1->data != h2->data) {

            return false;

        }

        h1 = h1->next;

        h2 = h2->next;

    }

    if (h1 != nullptr || h2 != nullptr) {

        return false;

    }

    return true;

}

int main() {

    int N;

    bool rez;

    Node\* head1 = nullptr;

    Node\* head2 = nullptr;

    cout<<"How many numbers you want to add in first list?  ";

    cin>>N;

    cout<<"Enter a first list:  ";

    for(int i=0; i<N; i++) {

        int k;

        cin>>k;

        add(head1, k);

    }

    cout<<"How many numbers you want to add in second list?  ";

    cin>>N;

    cout<<"Enter a second list:  ";

    for(int i=0; i<N; i++) {

        int k;

        cin>>k;

        add(head2, k);

    }

    rez=compare(head1, head2);

    if(rez) cout<<"The lists are the same";

    else cout<<"The list are different";

}

Перш за все оголошуємо бібліотеку iostream, Далі оголошується структура Node, що містить поля int data для зберігання цілочисельних значень та вказівник Node\* next, що вказує на наступний вузол у списку. Конструктор Node(int value) ініціалізує поле data переданим значенням та встановлює значення вказівника next як nullptr. Функція add додає новий вузол зі значенням value на початок списку, оновлюючи вказівник head. Функція compare порівнює два списки. Вона просувається одночасно по обох списках, порівнюючи значення в кожному вузлі. Якщо хоча б один з вузлів має відмінне значення або якщо один список закінчується раніше за інший, то списки вважаються різними.

У функції main спочатку вводиться кількість чисел для першого списку, потім вводяться самі числа для створення першого списку за допомогою функції add. Аналогічно створюється другий список. Після цього викликається функція compare, щоб порівняти обидва списки. Результат порівняння виводиться на екран, показуючи, чи списки ідентичні чи різні.

**Завдання №6 Сlass Practice Task 3**

#include<iostream>

#include<cmath>

using namespace std;

struct Node {

    int data;

    Node\* next;

    Node(int value) : data(value),next(nullptr) {}

};

void push(Node\* &head, int value){

    Node\* m = new Node(value);

    m->next = head;

    head = m;

}

int convert(Node\* h, int N){

    int rez = 0;

    Node\* current = h;

    for(int i = 0; i < N; i++){

        rez = rez \* 10 + current->data;

        current = current->next;

    }

    return rez;

}

int main(){

    int N,n, rez=0;

    Node\* head1 = nullptr;

    Node\* head2 = nullptr;

    cout<<"How many numbers you want to add in first list?  ";

    cin>>N;

    cout<<"Enter a first list:  ";

    for(int i=0; i<N; i++) {

        int k;

        cin>>k;

        push(head1, k);

    }

    cout<<"How many numbers you want to add in second list?  ";

    cin>>n;

    cout<<"Enter a second list:  ";

    for(int i=0; i<n; i++) {

        int k;

        cin>>k;

        push(head2, k);

    }

    rez=convert(head1, N)+convert(head2, n);

    cout<<rez;

}

Спочатку оголошуємо бібліотеки «iostreame» та «cmath», далі йде оголошення структури Node, яка містить поле int data для зберігання цілочисельних значень та вказівник Node\* next, який вказує на наступний вузол у списку. Конструктор Node(int value) ініціалізує поле data переданим значенням та встановлює значення вказівника next як nullptr. Функція push додає новий вузол зі значенням value на початок списку, оновлюючи вказівник head. Функція convert перетворює числа зі списку в ціле число. Вона перебирає елементи списку та побудоване число (значення зв'язаних вузлів) зберігає у змінній rez, використовуючи математичну операцію множення на 10 та додавання цифри відповідно до кожного вузла.

У функції main спочатку вводиться кількість чисел для першого списку, потім вводяться самі числа для створення першого списку за допомогою функції push. Аналогічно створюється другий список. Потім обидва числа перетворюються у ціле число за допомогою функції convert, сумуються і результат виводиться на екран.

**Завдання №7 Сlass Practice Task 4**

#include <iostream>

using namespace std;

struct Tree {

    int val;

    Tree\* left;

    Tree\* right;

    Tree(int value) : val(value), left(nullptr), right(nullptr) {}

};

void print(Tree\* root, int space=0) {

    int count = 2;

    if (root == nullptr) {

        return;

    }

    space += count;

    print(root->right, space);

    cout << endl;

    for (int i = count; i < space; i++) {

        cout << " ";

    }

    cout << root->val << "\n";

    print(root->left, space);

}

Tree\* create\_mirror\_flip(Tree\* root) {

    if (root == nullptr) {

        return nullptr;

    }

    Tree\* mirror = new Tree(root->val);

    mirror->left = create\_mirror\_flip(root->right);

    mirror->right = create\_mirror\_flip(root->left);

    return mirror;

}

int main() {

    Tree\* root = new Tree(5);

    root->left = new Tree(8);

    root->right = new Tree(1);

    root->left->left = new Tree(4);

    cout << "Entrance tree: " << endl;

    print(root);

    Tree\* mirrorRoot = create\_mirror\_flip(root);

    Tree\* mirroredTree = create\_mirror\_flip(root);

    cout << "Mirrored tree:" << endl;

    print(mirroredTree);

}

Першим кроком є під’єднання бібліотеки iostreame. Наступним етапом є оголошення структури Tree містить поле int val для зберігання значення вузла та вказівники Tree\* left та Tree\* right, які вказують на лівого та правого нащадка відповідно. Конструктор Tree(int value) ініціалізує поле val переданим значенням та встановлює вказівники left та right як nullptr. Функція print відображує дерево на екрані з урахуванням його структури. Вона використовує рекурсивний підхід для обходу дерева у прямому порядку, додаючи пробіли відповідно до глибини вузла, щоб наглядно показати його рівень у дереві. Функція create\_mirror\_flip створює дзеркальне відображення вхідного дерева. Вона також використовує рекурсивний підхід, створюючи нове дерево, де лівий нащадок оригінального дерева стає правим нащадком у дзеркальному дереві, а правий нащадок оригінального дерева стає лівим нащадком у дзеркальному дереві.

У функції main спочатку створюється вхідне дерево root з числами у вузлах. Потім викликається функція print, щоб відобразити це вхідне дерево. Наступною дією є створення дзеркального відображення mirroredTree за допомогою функції create\_mirror\_flip. Функція print знову викликається для відображення дзеркального дерева на екрані.

**Завдання №8 Сlass Practice Task 4**

#include <iostream>

using namespace std;

struct Tree{

    int v;

    Tree\* left;

    Tree\* right;

    Tree(int value) : v(value), left(NULL), right(NULL) {}

};

void print(Tree\* root, int s = 0) {

    int c = 3;

    if (root == NULL) {

        return;

    }

    s=s+c;

    print(root->right, s);

    cout << endl;

    for (int i = c; i < s; i++) {

        cout << " ";

    }

    cout << root->v << "\n";

    print(root->left, s);

}

int tree\_sum(Tree\* root){

    if ( root == NULL) {

        return 0;

    }

    if (root->right == NULL && root->left == NULL) {

        return root->v;

    }

    int ls= tree\_sum(root->left);

    int rs = tree\_sum(root->right);

    root->v = ls + rs;

    return root->v;

}

int main(){

    Tree\* root = new Tree(3);

    root->left = new Tree(10);

    root->right = new Tree(6);

    root->left->left = new Tree(8);

    root->left->right = new Tree(1);

    cout << "Before:" << endl;

    print(root);

    cout << endl;

    tree\_sum(root);

    cout << "After: " << endl;

    print(root);

    return 0;

}

Перш за все оголошуємо бібліотеку iostream. Структура Tree містить ціле значення int v для зберігання даних у вузлах, а також вказівники Tree\* left та Tree\* right, які вказують відповідно на лівого та правого нащадка вузла. Конструктор Tree(int value) ініціалізує поле v переданим значенням та встановлює вказівники left та right як NULL. Функція print використовує рекурсивний підхід для обходу дерева у прямому порядку, виводячи значення кожного вузла з відступом, що залежить від рівня вузла у дереві. Функція tree\_sum обчислює суму значень вузлів дерева. Вона також використовує рекурсивний підхід: якщо вузол є листовим (тобто, не має дочірніх вузлів), то повертається його власне значення. У протилежному випадку обчислюється сума значень лівого та правого нащадків, а потім ця сума присвоюється значенню поточного вузла.

У функції main спочатку створюється вхідне бінарне дерево root з числами у вузлах. Далі викликається функція print, щоб відобразити це дерево до зміни значень вузлів. Після цього викликається функція tree\_sum, яка обчислює суму значень вузлів та змінює значення кожного вузла на отриману суму. Нарешті, викликається функція print знову для відображення дерева після зміни значень вузлів.

**Завдання №9 Self Practice Task**

#include <iostream>

using namespace std;

struct Node {

    int data;

    Node\* left;

    Node\* right;

};

void print(Node\* root, int space=0) {

    int count = 2;

    if (root == nullptr) return;

    space += count;

    print(root->right, space);

    cout << endl;

    for (int i = count; i < space; i++) cout << " ";

    cout << root->data << "\n";

    print(root->left, space);

}

Node\* createNode(int value) {

    Node\* newNode = new Node();

    newNode->data = value;

    newNode->left = nullptr;

    newNode->right = nullptr;

    return newNode;

}

int getHeight(Node\* root) {

    if (root == nullptr) return 0;

    return max(getHeight(root->left), getHeight(root->right)) + 1;

}

bool isBalanced(Node\* root) {

    if (root == nullptr) return true;

    int leftHeight = getHeight(root->left);

    int rightHeight = getHeight(root->right);

    if (abs(leftHeight - rightHeight) <= 1 && isBalanced(root->left) && isBalanced(root->right)) return true;

    return false;

}

int main() {

    Node\* root = createNode(1);

    root->left = createNode(2);

    root->right = createNode(3);

    root->left->left = createNode(4);

    root->left->right = createNode(5);

    print(root);

    if (isBalanced(root)) {

        cout << "Tree is balanced" << endl;

    } else {

        cout << "Tree isn`t balanced" << endl;

    }

    return 0;

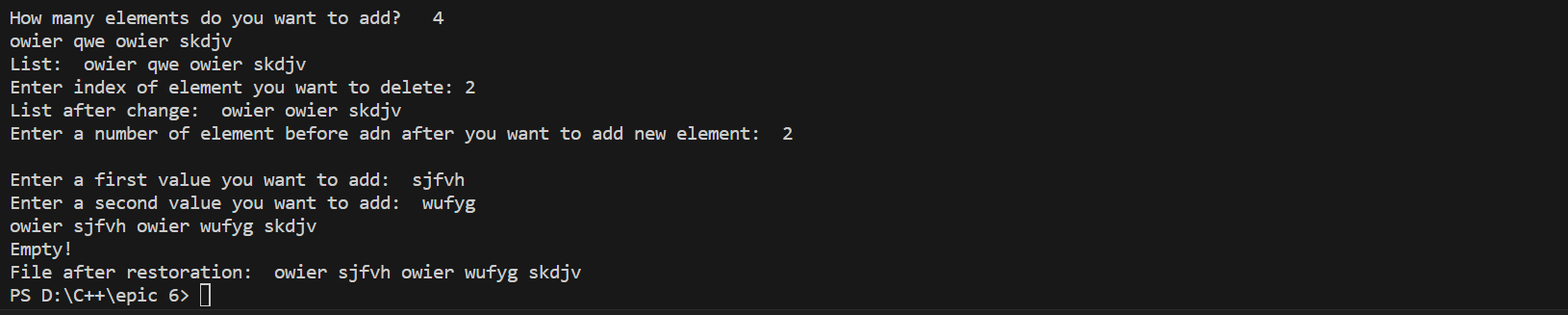
}

Перш за все оголошується бібліотека «iostreame». Після, Структура Node , що представляє вузол бінарного дерева. У кожного вузла є поле data, яке містить дані, а також вказівники left та right, що вказують на лівого та правого дочірніх вузла відповідно. Функція print виводить бінарне дерево у вигляді горизонтальної структури. Вона приймає вузол root та параметр space (початковий пробіл). Функція використовує рекурсію для обходу дерева в порядку "правий вузол, поточний вузол, лівий вузол". Для кожного вузла виводиться його значення з пробілами для створення відступу від лівого краю екрану. Функція createNode створює новий вузол з переданим значенням. Функція getHeight рекурсивно обчислює висоту піддерева, починаючи від заданого вузла. Функція isBalanced перевіряє бінарне дерево на баланс. Вона рекурсивно обчислює висоти лівого та правого піддерева для кожного вузла та перевіряє, чи вони відрізняються більше, ніж на одиницю. Якщо так, то дерево вважається незбалансованим.

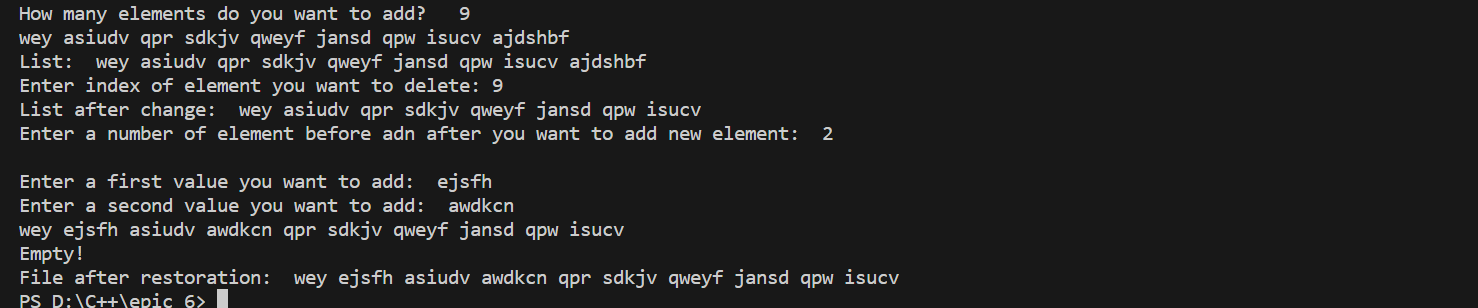
Далі, у функції main створюється бінарне дерево з вузлів, виводиться його структура за допомогою функції print, а потім викликається функція isBalanced, щоб перевірити, чи є дерево збалансованим, та вивести відповідне повідомлення.

## **4. Результати виконання завдань, тестування та фактично затрачений час:**

**Завдання №1 VNS Lab 10**

****

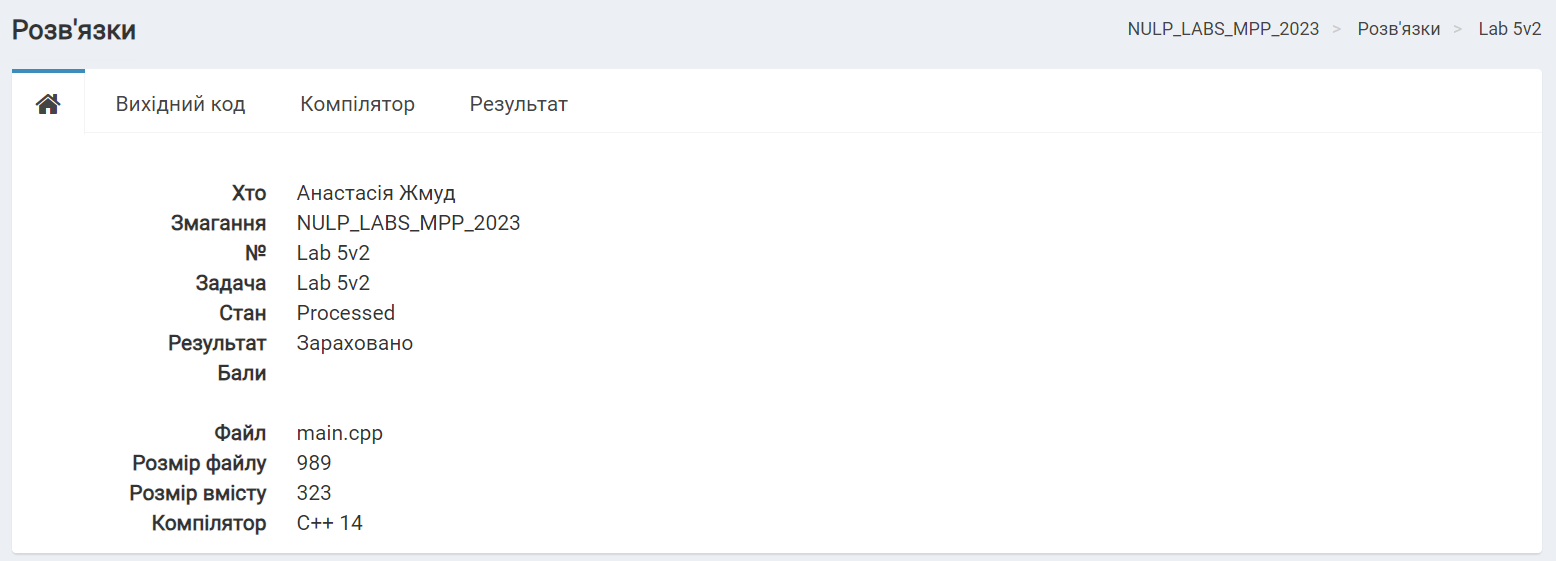
**Тест 1**

****

**Тест 2**

Час затрачений на виконання завдання: 6 годин

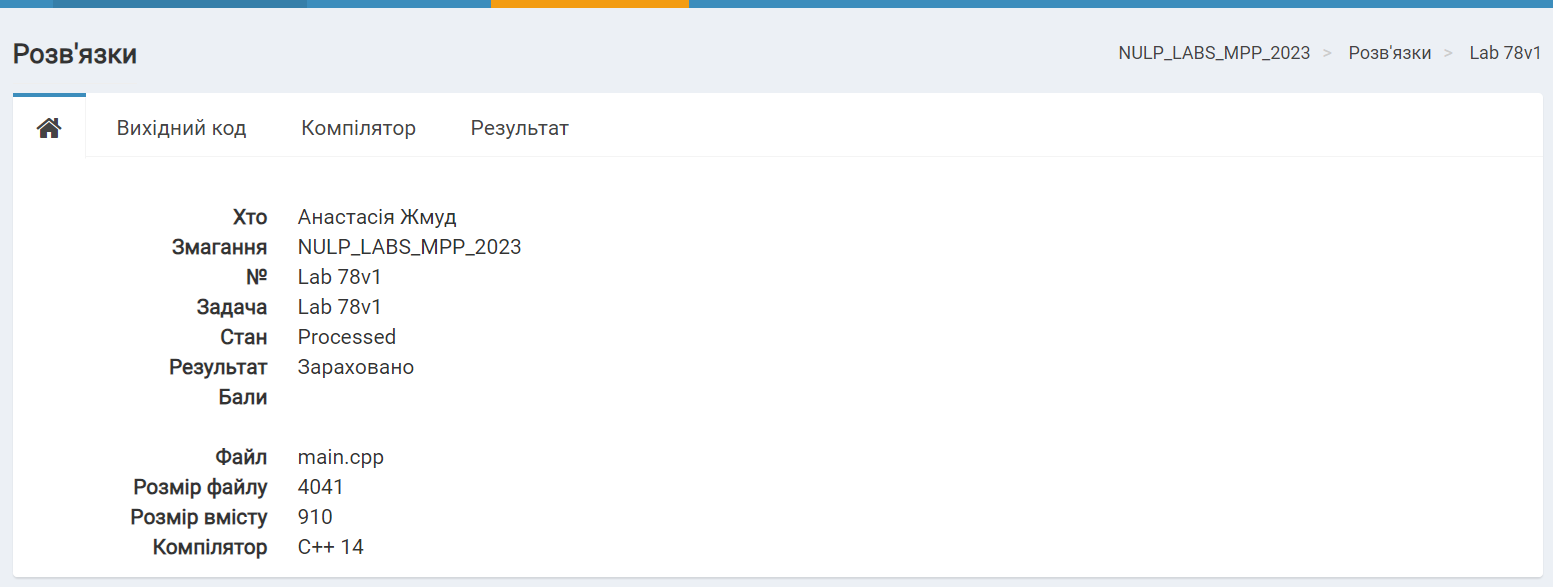
**Завдання №2 Algotester Lab 5**

****

**Зарахування програми на Algotester**

Час затрачений на виконання завдання: 2 години

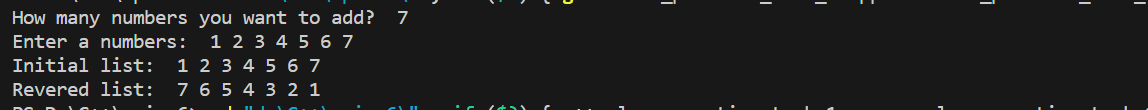
**Завдання №3 Algotester Lab 7-8**

****

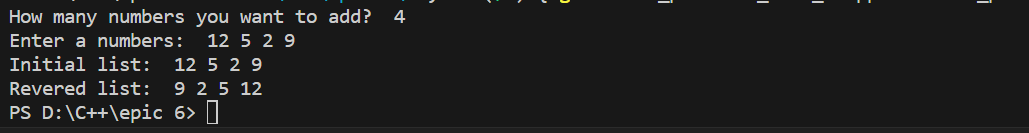
**Зарахування програми на Algotester**

Час затрачений на виконання завдання: 8 годин

**Завдання №4 Class Practice Task 1**

****

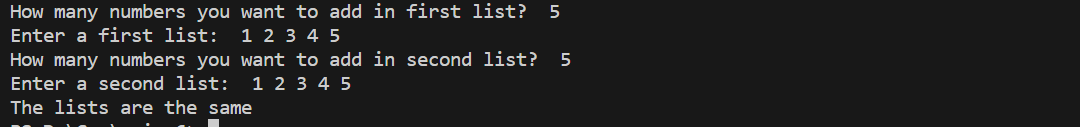
**Тест 1**

****

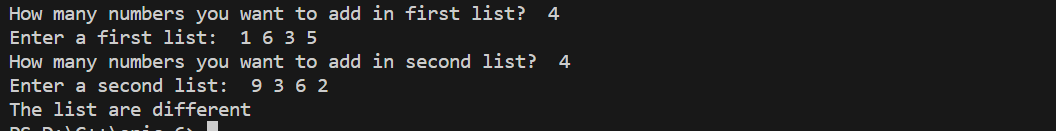
**Тест 2**

Час затрачений на виконання завдання: 30 хвилин

**Завдання №5 Class Practice Task 2**

****

**Тест 1**

****

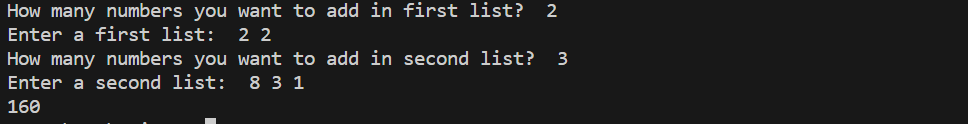
**Тест 2**

Час затрачений на виконання завдання: 30 хвилин

**Завдання №6 Class Practice Task 3**

****

**Тест 1**

****

**Тест 2**

Час затрачений на виконання завдання: 30 хвилин

**Завдання №7 Class Practice Task 4**

****

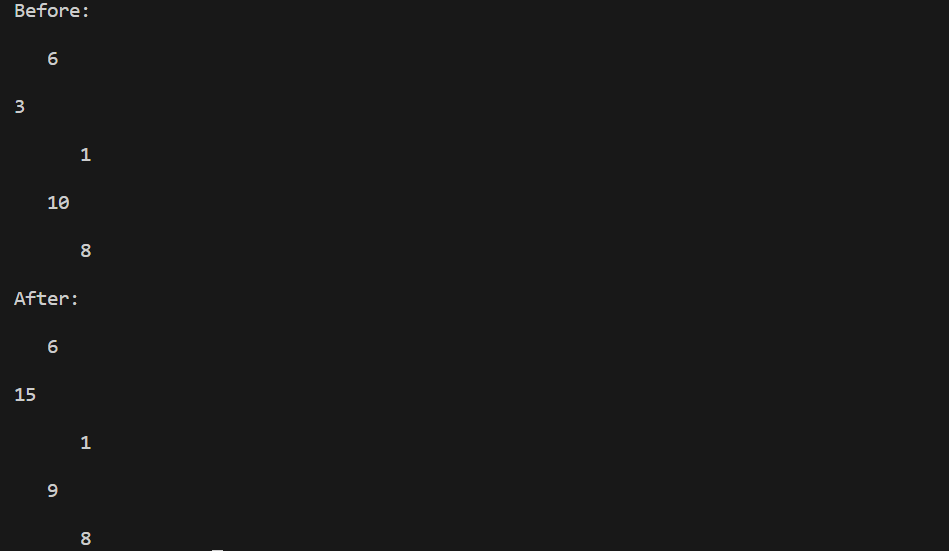
**Тест 1**

****

**Тест 2**

Час затрачений на виконання завдання: 1 година

**Завдання №8 Class Practice Task 5**

****

**Тест 1**

****

**Тест 2**

Час затрачений на виконання завдання: 1 година

**Завдання №9 Self Practice Task**

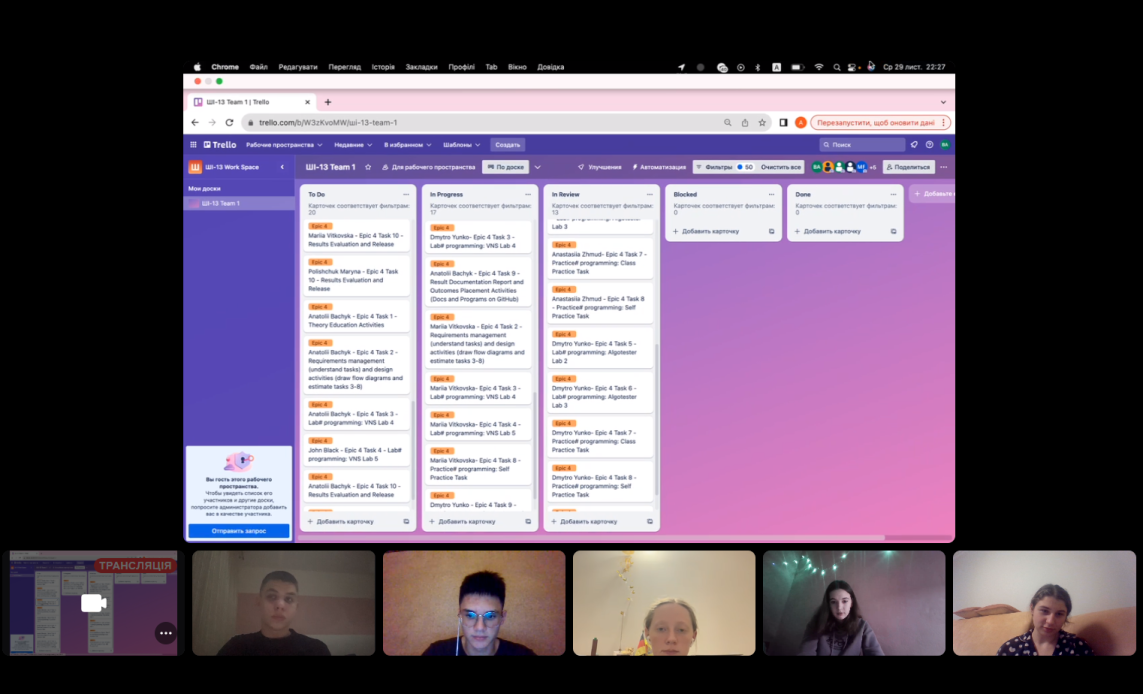
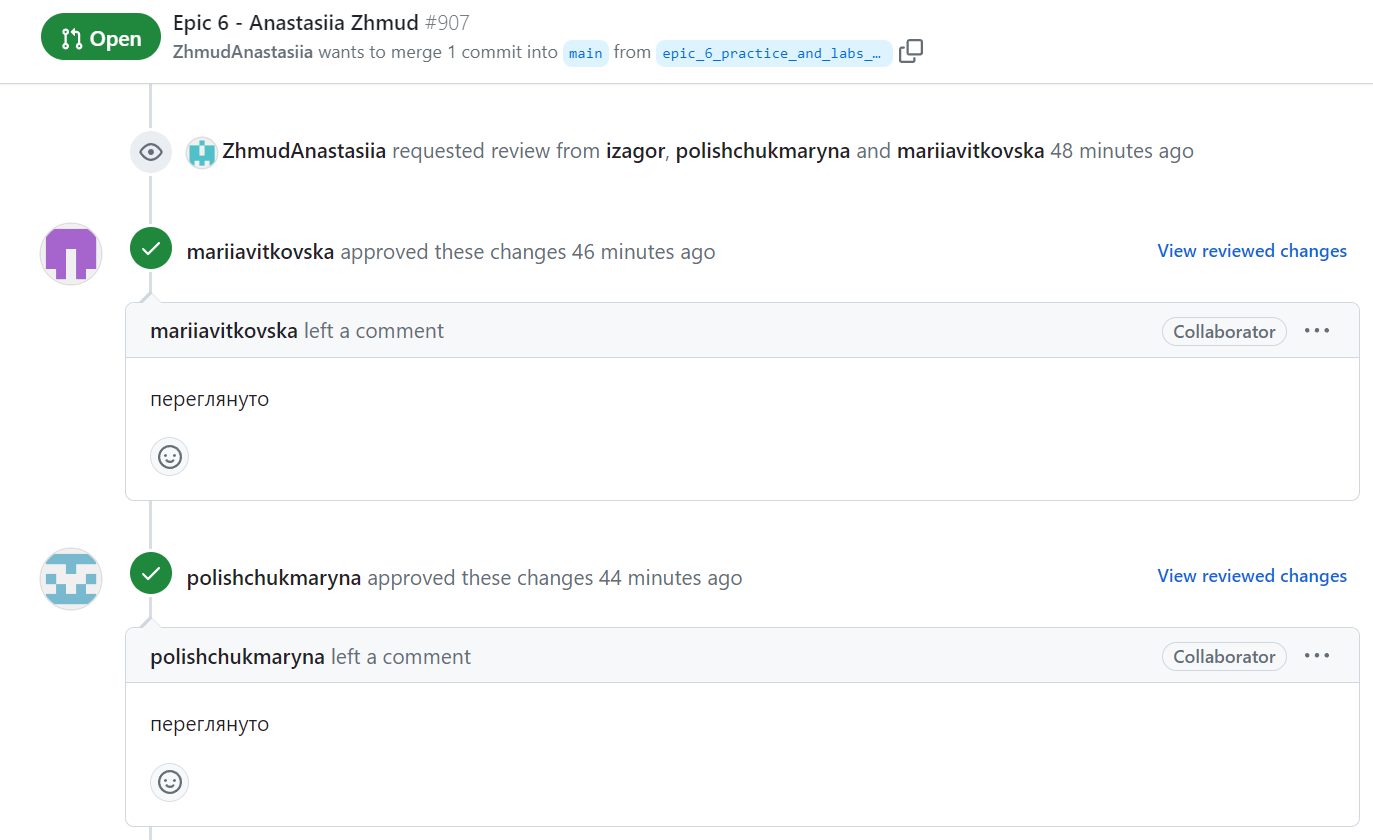
****

**Тест 1**

****

**Тест 2**

## **5. Кооперація з командою:**

* Скрін з зустрічі по обговоренню задач Епіку та Скрін прогресу по Трелло:
* ****Скрін з 2-му коментарями від учасників команди на пул реквесті з Ревю Роботи:

# **Висновки:**

Виконуючи лабораторні і практичні роботи №6 детально ознайомилися з різними динамічними структурами даних, такими як одно- та двонаправлені списки, а також бінарні дерева. Вивчаючи ці структури, я здобула навички оптимізації обробки даних. Я можу оцінювати, яка структура буде оптимальною для конкретних завдань на основі швидкодії операцій. Окрім списків, Я також отримала досвід роботи з бінарними деревами. Я розумію властивості бінарних дерев та їхній потенціал для швидкодії пошуку, вставки та видалення даних.