Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту



**Звіт**

**про виконання лабораторних та практичних робіт блоку № 6**

На тему:  «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.»

***з дисципліни:*** «Основи програмування»

до:

Практичних Робіт до блоку № 6

**Виконала:**

Студентка групи ШІ-12

Лазаревич Юлія Дмитрівна

Львів 2024

**Тема роботи:**

Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.

**Мета роботи:**

Ознайомитись з динамічними структурами (Чергою, Стеком, Списком, Деревом) та алгоритмом обробки динамічних структур.

**Теоретичні відомості:**

Тема №1: Основи Динамічних Структур Даних.

* Джерела Інформації:
* Лекції О. Пшеничного
* Практичні М. Фаріон
* Що опрацьовано:
* Динамічні структури даних дозволяють гнучко змінювати розмір і вміст, залежно від потреб програми. Вони працюють із пам'яттю в купі (heap), а не на стеку, забезпечуючи ефективне використання ресурсів.
* У C++ пам'ять виділяється для стеку автоматично, а для хіпу — вручну за допомогою new та звільняється delete. Наприклад:

int\* p = new int(5); // виділення пам'яті в купі

delete p; // звільнення пам'яті

* Приклади простих динамічних структур: динамічний масив. Клас std::vector є прикладом динамічного масиву в C++. Його можна змінювати, не знаючи заздалегідь розмір:

std::vector<int> vec = {1, 2, 3};

vec.push\_back(4); // Додаємо елемент

* Статус: ознайомлена.
* Початок опрацювання теми: 22.11.24
* Завершення опрацювання теми 26.11.24

Тема №2: Стек.

* Джерела Інформації:
* Лекції О. Пшеничного
* Практичні М. Фаріон
* Що опрацьовано:
* Стек — це структура, яка працює за принципом "останній прийшов — перший вийшов" (LIFO).
* Операції push, pop, top: реалізація та використання. Операції реалізуються вручну або з використанням std::stack. Наприклад:

std::stack<int> s;

s.push(10); // додаємо

s.pop(); // видаляємо

int top = s.top(); // отримуємо останній елемент

* Приклади використання стеку: обернений польський запис, перевірка балансу дужок. Для перевірки дужок використовують стек для збереження відкритих дужок і перевірки закритих.
* Початок опрацювання теми: 22.11.24
* Завершення опрацювання теми 25.11.24

Тема №3: Черга.

* Джерела Інформації:
* [Черга. Способи реалізації черги.](https://www.bestprog.net/uk/2019/09/26/c-queue-general-concepts-ways-to-implement-the-queue-implementing-a-queue-as-a-dynamic-array-ua/)
* Що опрацьовано:
* Визначення та властивості черги. Черга працює за принципом "перший прийшов — перший вийшов" (FIFO).
* Операції enqueue, dequeue, front. У C++ чергу можна реалізувати за допомогою std::queue:

std::queue<int> q;

q.push(1); // enqueue

q.pop(); // dequeue

int front = q.front(); // отримуємо перший елемент

* Пріоритетна черга. Пріоритетні черги реалізують чергу, де елементи обробляються за їх важливістю, використовуючи std::priority\_queue [7][8][9].
* Статус: ознайомлена.
* Початок опрацювання теми: 27.11.24
* Завершення опрацювання теми 28.11.24

Тема №4: Зв'язні Списки.

* Джерела Інформації:
* [Лінійний однозв’язний список. Загальні відомості. Базові операції над списком](https://www.bestprog.net/uk/2022/02/11/c-linear-singly-linked-list-general-information-ua/)
* Що опрацьовано:
* Однозв'язний і двозв'язний списки. Однозв'язний список містить вузли, кожен із яких вказує на наступний. У двозв'язному списку вузли мають покажчики на попередній і наступний елемент.
* Операції: обхід, пошук, видалення. Для створення списку потрібно динамічно виділяти пам'ять під вузли, наприклад:

struct Node {

int data;

Node\* next;

};

* Серіалізація: використовується для збереження складних структур у файл і відновлення їх.
* Статус: ознайомлена.
* Початок опрацювання теми: 28.11.24
* Завершення опрацювання теми 29.11.24

Тема №5: Дерева.

* Джерела Інформації:
* Лекції О. Пшеничного
* Практичні М. Фаріон
* Що опрацьовано:
* Дерево — ієрархічна структура даних. Бінарні дерева мають вузли з максимум двома дочірніми елементами.
* Обхід дерева: в глибину і ширину. Рекурсивний обхід у глибину:

void inorder(Node\* root) {

if (root) {

inorder(root->left);

std::cout << root->data << " ";

inorder(root->right);

}

}

* AVL і червоно-чорні дерева. AVL — це дерево, де різниця висот піддерев кожного вузла не перевищує 1. Червоно-чорні дерева підтримують балансування шляхом кольорового маркування вузлів.
* Статус: ознайомлена.
* Початок опрацювання теми: 28.11.24
* Завершення опрацювання теми 29.11.24

Тема №6: Алгоритми Обробки Динамічних Структур.

* Джерела Інформації:
* Лекції О. Пшеничного
* Практичні М. Фаріон
* Що опрацьовано:
* Основи алгоритмічних патернів: ітеративні, рекурсивні. Ітеративний підхід використовує цикли, тоді як рекурсивний викликає функцію повторно.
* Алгоритми пошуку та сортування. Наприклад, для списків використовують лінійний пошук або вставки.
* Статус: ознайомлена.
* Початок опрацювання теми: 22.11.24
* Завершення опрацювання теми 23.11.24

**Виконання роботи:**

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

Завдання №1 – VNS Lab 10 – Variant 10.

Написати програму, у якій створюються динамічні структури й виконати

їхню обробку у відповідності зі своїм варіантом.

*Для кожного варіанту розробити такі функції:*

1. Створення списку.

2. Додавання елемента в список (у відповідності зі своїм варіантом).

3. Знищення елемента зі списку (у відповідності зі своїм варіантом).

4. Друк списку.

5. Запис списку у файл.

6. Знищення списку.

7. Відновлення списку з файлу.

*Порядок виконання роботи*

1. Написати функцію для створення списку. Функція може створювати

порожній список, а потім додавати в нього елементи.

2. Написати функцію для друку списку. Функція повинна передбачати вивід

повідомлення, якщо список порожній.

3. Написати функції для знищення й додавання елементів списку у

відповідності зі своїм варіантом.

4. Виконати зміни в списку й друк списку після кожної зміни.

5. Написати функцію для запису списку у файл.

6. Написати функцію для знищення списку.

7. Записати список у файл, знищити його й виконати друк (при друці повинне

бути видане повідомлення "Список порожній").

8. Написати функцію для відновлення списку з файлу.

9. Відновити список і роздрукувати його.

10.Знищити список.

«Записи в лінійному списку містять ключове поле типу int. Сформувати

двонаправлений список. Додати в нього елемент із заданим номером,

знищити К елементів з кінця списку.»

Завдання №2 – Algotester Lab 5 – Variant 1.

У світі Атод сестри Ліна і Рілай люблять грати у гру. У них є дошка із 8-ми рядків і 8-ми стовпців. На перетині i-го рядкa і j-го стовпця лежить магічна куля, яка може світитись магічним світлом (тобто у них є 64 кулі). На початку гри деякі кулі світяться, а деякі ні... Далі вони обирають NN куль і для кожної читають магічне заклинання, після чого всі кулі, які лежать на перетині стовпця і рядка обраної кулі міняють свій стан (ті що світяться - гаснуть, ті, що не світяться - загораються).

Також вони вирішили трохи Вам допомогти і придумали спосіб як записати стан дошки одним числом a із 8-ми байт, а саме (див. Примітки):

* Молодший байт задає перший рядок матриці;
* Молодший біт задає перший стовпець рядку;
* Значення біту каже світиться куля чи ні (0 - ні, 1 - так);

Тепер їх цікавить яким буде стан дошки після виконання N заклинань і вони дуже просять Вас їм допомогти.

# Вхідні дані

У першому рядку одне число a - поточний стан дошки.

У другому рядку NN - кількість заклинань.

У наступних NN рядках по 2 числа Ri, Ci - рядок і стовпець кулі над якою виконується заклинання.

# Вихідні дані

Одне число b - стан дошки після виконання N заклинань.

Завдання №3 – Algotester Lab 7-8 – Variant 2.

Ваше завдання - власноруч реалізувати структуру даних "Динамічний масив".  
Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його аргументи.  
  
Вам будуть поступати запити такого типу:

* **Вставка**:  
  Ідентифікатор - insert  
  Ви отримуєте ціле число index елемента, на місце якого робити вставку.  
  Після цього в наступному рядку написане число NN - розмір масиву, який треба вставити.  
  У третьому рядку NN цілих чисел - масив, який треба вставити на позицію index.
* **Видалення**:  
  Ідентифікатор - eras  
  Ви отримуєте 2 цілих числа - index, індекс елемента, з якого почати видалення таn - кількість елементів, яку треба видалити.
* **Визначення розміру**:  
  Ідентифікатор - size  
  Ви не отримуєте аргументів.  
  Ви виводите кількість елементів у динамічному масиві.
* **Визначення кількості зарезервованої пам’яті**:  
  Ідентифікатор - capacit  
  Ви не отримуєте аргументів.  
  Ви виводите кількість зарезервованої пам’яті у динамічному масиві.  
  Ваша реалізація динамічного масиву має мати фактор росту ([Growth factor](https://en.wikipedia.org/wiki/Dynamic_array" \l "Growth_factor)) рівний 2.
* **Отримання значення** ii-го елементу  
  Ідентифікатор - ge  
  Ви отримуєте ціле число - index, індекс елемента.  
  Ви виводите значення елемента за індексом. Реалізувати використовуючи перегрузку оператора [][]
* **Модифікація значення** ii-го елементу  
  Ідентифікатор - set  
  Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення. Реалізувати використовуючи перегрузку оператора [][]
* **Вивід динамічного масиву на екран**  
  Ідентифікатор - print  
  Ви не отримуєте аргументів.  
  Ви виводите усі елементи динамічного масиву через пробіл.  
  Реалізувати використовуючи перегрузку оператора <<<<

**Вхідні дані**

Ціле число Q - кількість запитів.  
У наступних рядках Q запитів у зазначеному в умові форматі.

**Вихідні дані**

Відповіді на запити у зазначеному в умові форматі.

Завдання №4 – Algotester Lab 7-8 – Variant 1.

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.  
  
Вам будуть поступати запити такого типу:

* **Вставка**:  
  Ідентифікатор - insert  
  Ви отримуєте ціле число index елемента, на місце якого робити вставку.  
  Після цього в наступному рядку написане число N - розмір списку, який треба вставити.  
  У третьому рядку N цілих чисел - список, який треба вставити на позицію index.
* **Видалення**:  
  Ідентифікатор - erasee  
  Ви отримуєте 2 цілих числа - index, індекс елемента, з якого почати видалення та n - кількість елементів, яку треба видалити.
* **Визначення розміру**:  
  Ідентифікатор - size  
  Ви не отримуєте аргументів.  
  Ви виводите кількість елементів у списку.
* **Отримання значення** i-го елементу  
  Ідентифікатор - get  
  Ви отримуєте ціле число - index, індекс елемента.  
  Ви виводите значення елемента за індексом.
* **Модифікація значення** ii-го елементу  
  Ідентифікатор - set  
  Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення.
* **Вивід списку на екран**  
  Ідентифікатор - print  
  Ви не отримуєте аргументів.  
  Ви виводите усі елементи списку через пробіл.  
  Реалізувати використовуючи перегрузку оператора <<<<

**Вхідні дані**

Ціле число Q - кількість запитів.  
У наступних рядках Q запитів у зазначеному в умові форматі.

**Вихідні дані**

Відповіді на запити у зазначеному в умові форматі.

Завдання №6**–** Class Practice Work - Зв’язаний список.

**Задача №1 - Реверс списку (Reverse list)**

***Реалізувати метод реверсу списку:*** Node\* reverse(Node \*head);

*Умови задачі:*

-       використовувати цілочисельні значення в списку;

-       реалізувати метод реверсу;

-       реалізувати допоміжний метод виведення вхідного і обернутого списків;

**Мета задачі**

***Розуміння структур даних:*** Реалізація методу реверсу для зв’язаних списків є чудовим способом для поглиблення розуміння зв’язаних списків як фундаментальної структури даних. Він заохочує практичний підхід до вивчення того, як структуруються пов’язані списки та як ними маніпулювати.

***Розвиток алгоритмічне мислення:*** Це завдання розвиває алгоритмічне мислення. Перевертання пов’язаного списку вимагає логічного підходу до маніпулювання покажчиками, що є ключовим навиком у інформатиці.

***Засвоїти механізми маніпуляції з покажчиками:*** пов’язані списки значною мірою залежать від покажчиків. Це завдання покращить навички маніпулювання вказівниками, що є ключовим аспектом у таких мовах, як C++.

***Розвинути навички розв’язувати задачі:*** перевернути пов’язаний список непросто й вимагає творчого й логічного мислення, таким чином покращуючи свої навички розв’язування поставлених задач.

**Пояснення прикладу**

Спочатку ми визначаємо просту структуру ***Node*** для нашого пов’язаного списку.

Потім функція ***reverse*** ітеративно змінює список, маніпулюючи наступними покажчиками кожного вузла.

***printList*** — допоміжна функція для відображення списку.

Основна функція створює зразок списку, демонструє реверсування та друкує вихідний і обернений списки.

**Задача №2 - Порівняння списків**

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

*Умови задачі:*

-       використовувати цілочисельні значення в списку;

-       реалізувати функцію, яка ітеративно проходиться по обох списках і порівнює дані в кожному вузлі;

-       якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає ***false***.

**Мета задачі**

***Розуміння рівності в структурах даних:*** це завдання допомагає зрозуміти, як визначається рівність у складних структурах даних, таких як зв’язані списки. На відміну від примітивних типів даних, рівність пов’язаного списку передбачає порівняння кожного елемента та їх порядку.

***Поглиблення розуміння зв’язаних списків:*** Порівнюючи зв’язані списки, дозволяють покращити своє розуміння обходу, фундаментальної операції в обробці зв’язаних списків.

***Розуміння ефективність алгоритму:*** це завдання також вводить поняття ефективності алгоритму. Студенти вчаться ефективно порівнювати елементи, що є навичкою, важливою для оптимізації та зменшення складності обчислень.

***Розвинути базові навики роботи з реальними програми:*** функції порівняння мають вирішальне значення в багатьох реальних програмах, таких як виявлення змін у даних, синхронізація структур даних або навіть у таких алгоритмах, як сортування та пошук.

***Розвинути навик вирішення проблем і увага до деталей:*** це завдання заохочує скрупульозний підхід до програмування, оскільки навіть найменша неуважність може призвести до неправильних результатів порівняння. Це покращує навички вирішення проблем і увагу до деталей.

**Пояснення прикладу**

●      Для пов’язаного списку визначено структуру ***Node***.

●      Функція ***compare*** ітеративно проходить обидва списки одночасно, порівнюючи дані в кожному вузлі.

●      Якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає ***false***.

●      Основна функція ***main*** створює два списки та демонструє порівняння.

**Задача №3 – Додавання великих чисел**

Node\* add(Node \*n1, Node \*n2);

*Умови задачі:*

-       використовувати цифри від 0 до 9 для значень у списку;

-       реалізувати функцію, яка обчислює суму двох чисел, які збережено в списку; молодший розряд числа записано в голові списка (напр. 379  ⟹  9→7→3);

-       функція повертає новий список, передані в функцію списки не модифікуються.

**Мета задачі**

***Розуміння операцій зі структурами даних:*** це завдання унаочнює практичне використання списка для обчислювальних потреб. Арифметичні операції з великими числами це окремий клас задач, для якого використання списків допомагає обійти обмеження у представленні цілого числа сучасними комп’ютерами.

***Поглиблення розуміння зв’язаних списків:*** Застосовування зв’язаних списків для арифметичних операції з великими числами дозволяє покращити розуміння операцій з обробки зв’язаних списків.

***Розуміння ефективність алгоритму:*** це завдання дозволяє порівняти швидкість алгоритму додавання з використанням списків зі швидкістю вбудованих арифметичних операцій. Студенти вчаться розрізняти позитивні та негативні ефекти при виборі структур даних для реалізації практичних програм.

***Розвинути базові навики роботи з реальними програми:*** арифметичні операції з великими числами використовуються у криптографії, теорії чисел, астрономії, та ін.

***Розвинути навик вирішення проблем і увага до деталей:*** завдання покращує розуміння обмежень у представленні цілого числа сучасними комп’ютерами та пропонує спосіб його вирішення.

**Бінарні дерева**

**Задача №4 - Віддзеркалення дерева**

TreeNode \*create\_mirror\_flip(TreeNode \*root);

*Умови задачі:*

-       використовувати цілі числа для значень у вузлах дерева

-       реалізувати функцію, що проходить по всіх вузлах дерева і міняє місцями праву і ліву вітки дерева

-       функція повертає нове дерево, передане в функцію дерево не модифікується

**Мета задачі**

***Розуміння структур даних:*** Реалізація методу віддзеркалення бінарного дерева покращує розуміння структури бінарного дерева, виділення пам’яті для вузлів та зв’язування їх у єдине ціле. Це один з багатьох методів роботи з бінарними деревами.

***Розвиток алгоритмічне мислення:*** Це завдання розвиває алгоритмічне мислення. Прохід всіх вузлів дерева продемонструє розгортання рекурсивного виклику.

**Задача №5 - Записати кожному батьківському вузлу суму підвузлів**

void tree\_sum(TreeNode \*root);

*Умови задачі:*

-       використовувати цілочисельні значення у вузлах дерева;

-       реалізувати функцію, яка ітеративно проходить по бінарному дереві і записує у батьківський вузол суму значень підвузлів

-       вузол-листок не змінює значення

-       значення змінюються від листків до кореня дерева

**Мета задачі**

***Розуміння структур даних:*** Реалізація методу підрахунку сум підвузлів бінарного дерева покращує розуміння структури бінарного дерева. Це один з багатьох методів роботи з бінарними деревами.

***Розвиток алгоритмічне мислення:*** Це завдання розвиває алгоритмічне мислення. Прохід всіх вузлів дерева демонструє розгортання рекурсивного виклику.

Завдання №7- Self Practice Work – Algotester Lab 5 – Variant 2.

У вас є карта гори розміром N×M.

Також ви знаєте координати {x,y}, у яких знаходиться вершина гори.

Ваше завдання - розмалювати карту таким чином, щоб найнижча точка мала число 0, а пік гори мав найбільше число.

Клітинкі які мають суміжну сторону з вершиною мають висоту на один меншу, суміжні з ними і не розфарбовані мають ще на 1 меншу висоту і так далі.

# Вхідні дані

У першому рядку 2 числа N та M - розміри карти

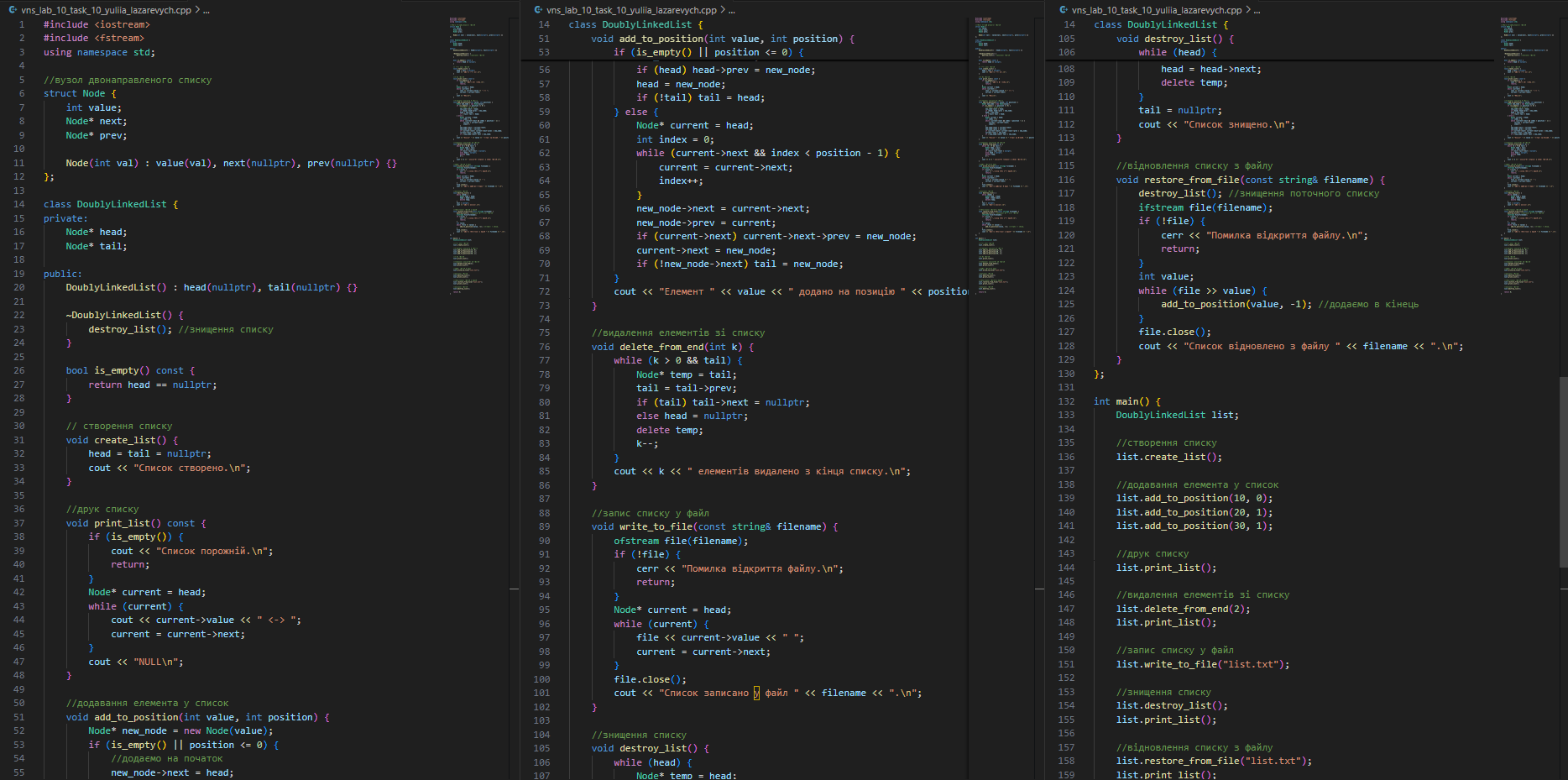
у другому рядку 2 числа x та y - координати піку гори

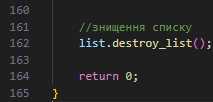
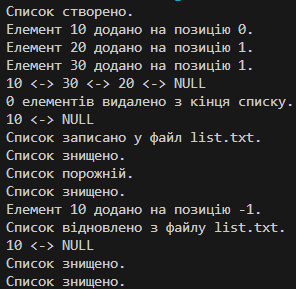
# Вихідні дані

N рядків по M елементів в рядку через пробіл - висоти карти.

1. **Код програм з посиланням на зовнішні ресурси:**

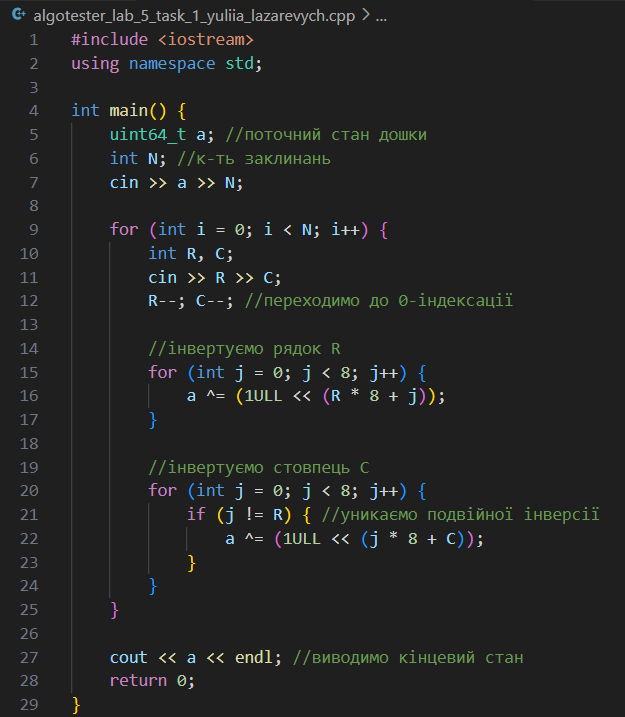
Завдання №1 – VNS Lab 10 – Variant 10.

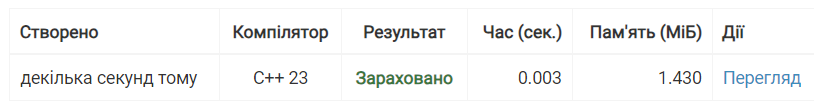




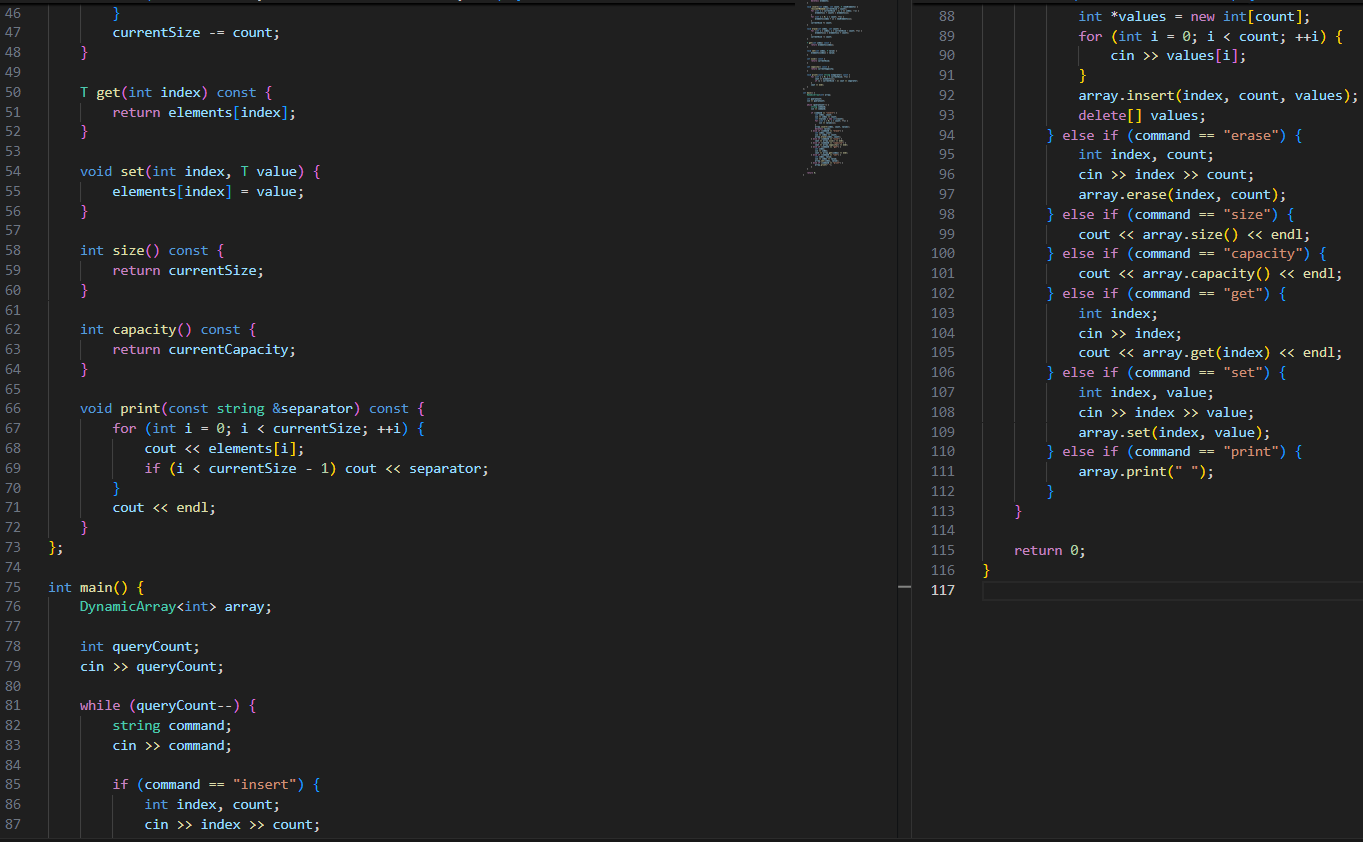
Завдання №2 – Algotester Lab 5 – Variant 1.

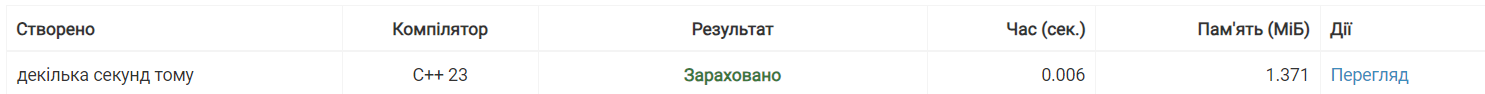




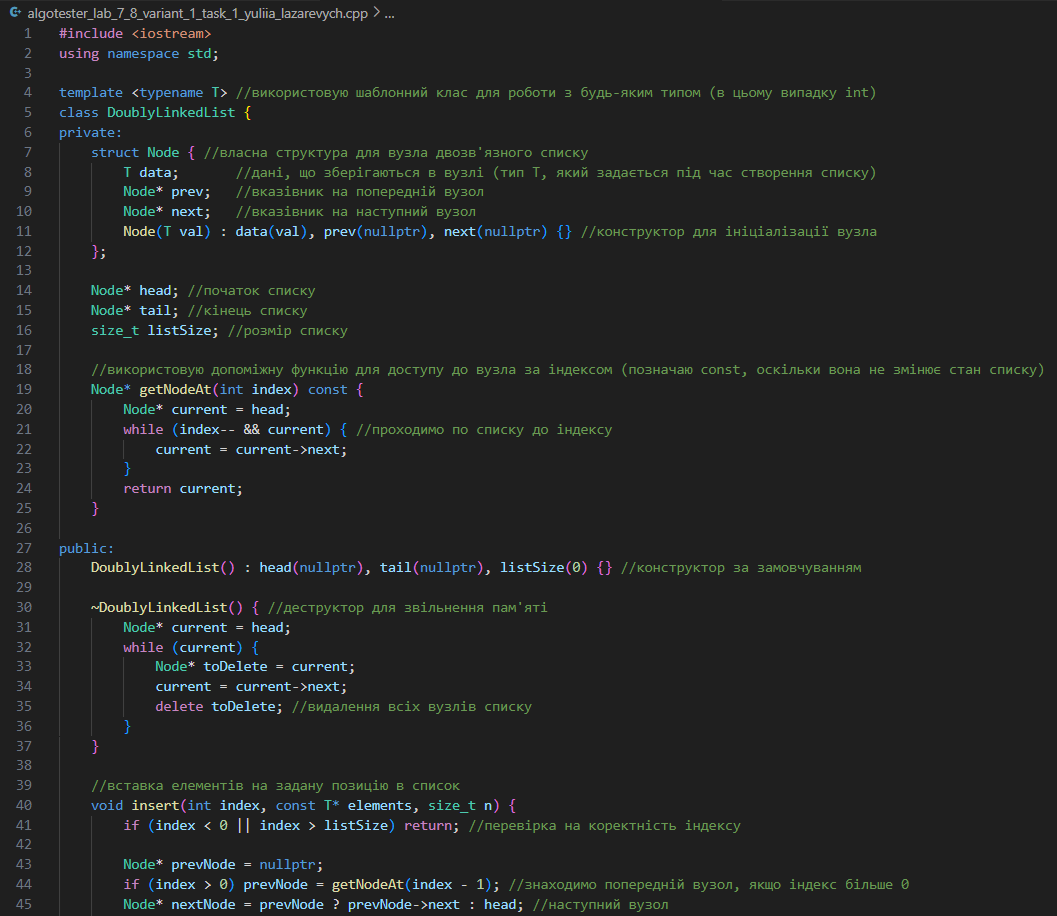
Завдання №3 – Algotester Lab 7-8 – Variant 2.





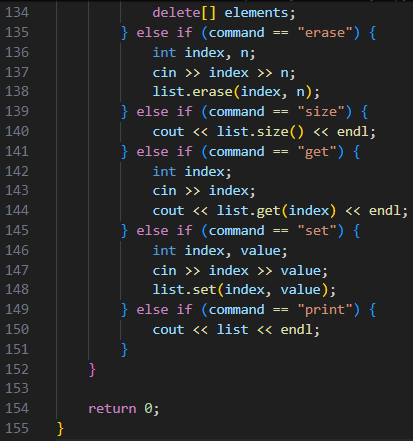


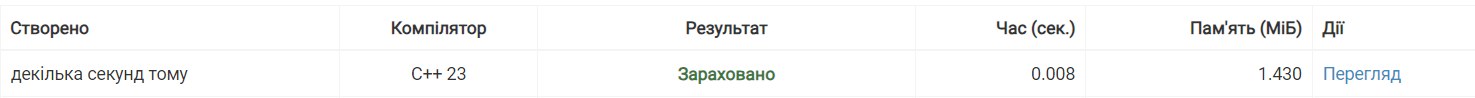
Завдання №4 – Algotester Lab 7-8 – Variant 1.



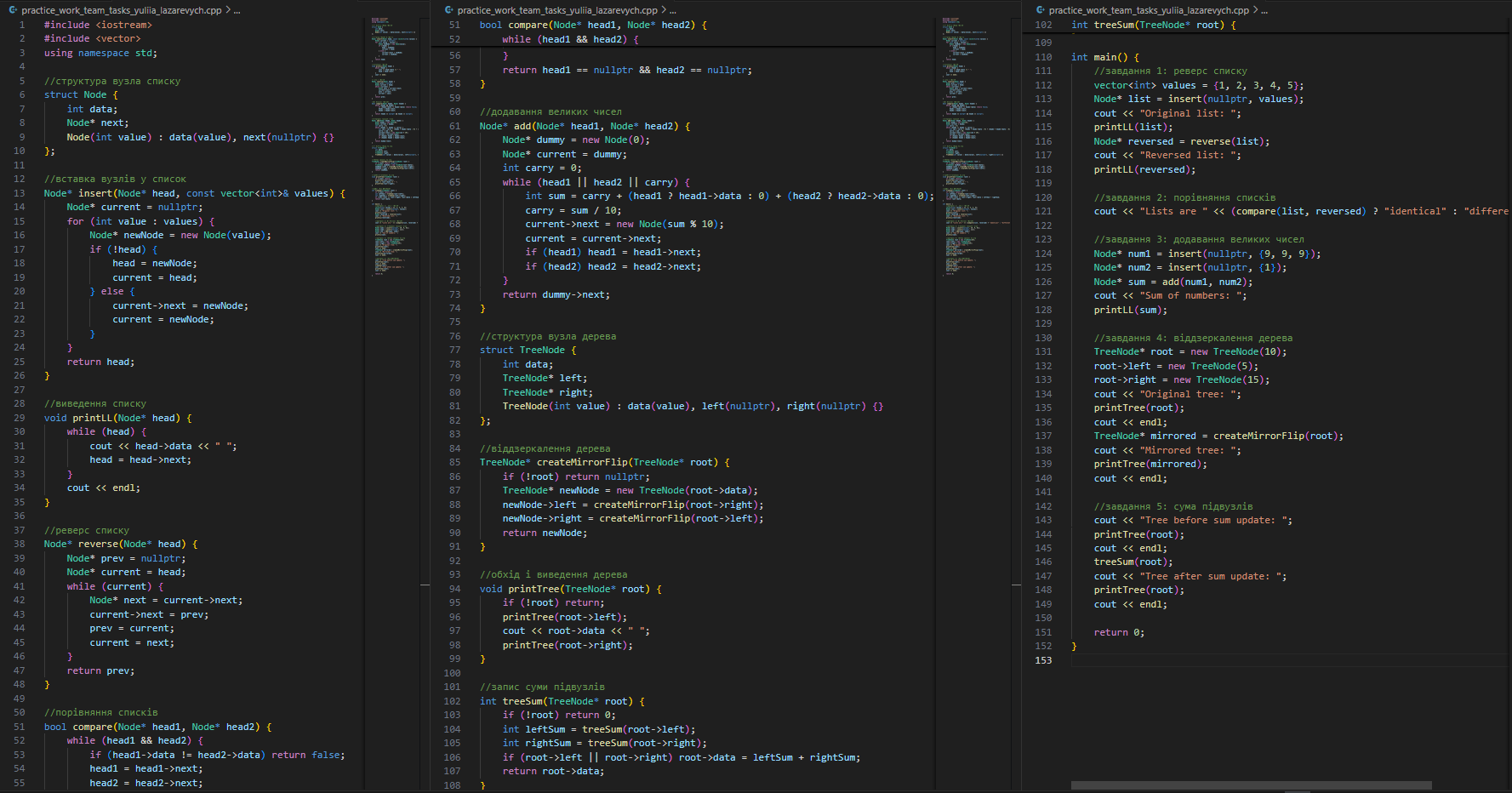


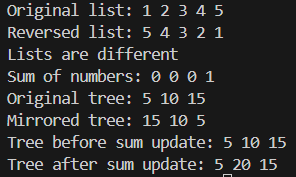




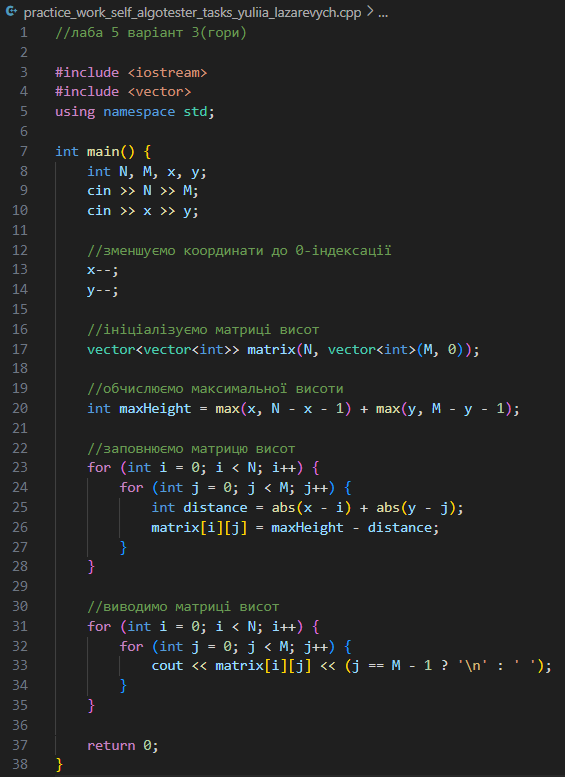


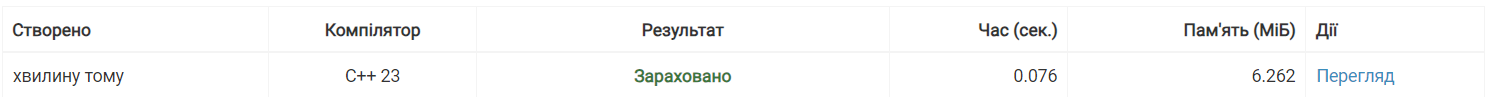
Завдання №6**–** Class Practice Work - Зв’язаний список.





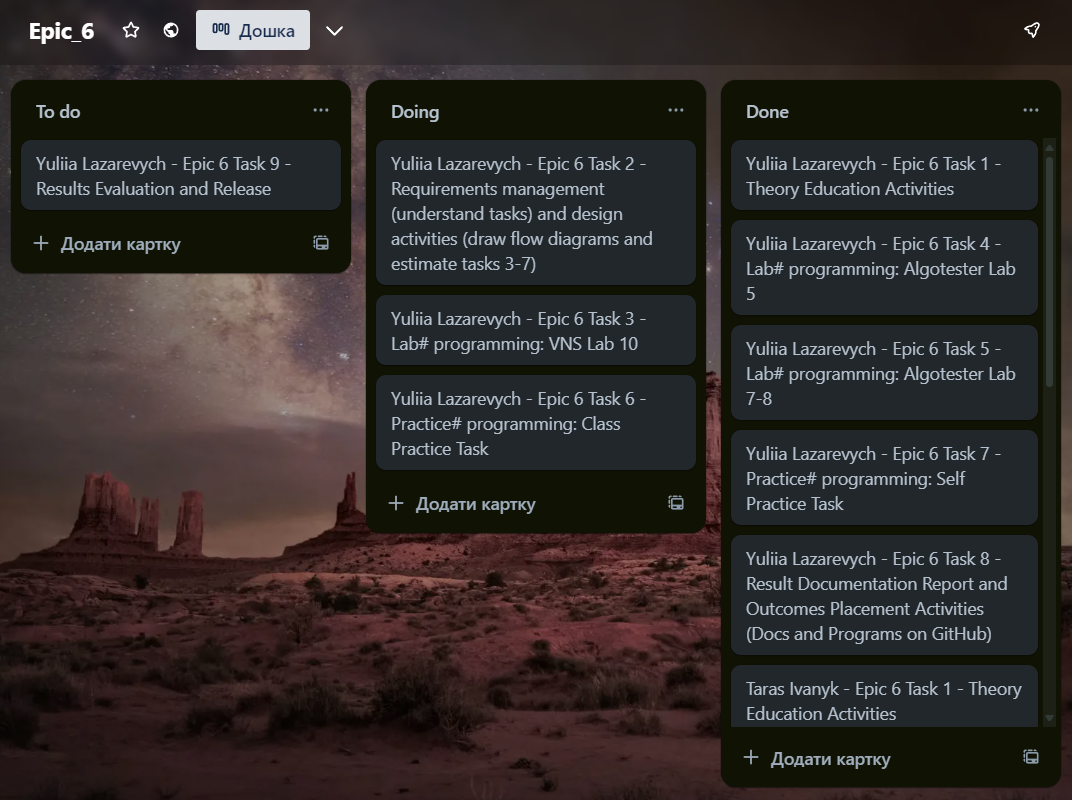
Завдання №7- Self Practice Work – Algotester Lab 5 – Variant 2.





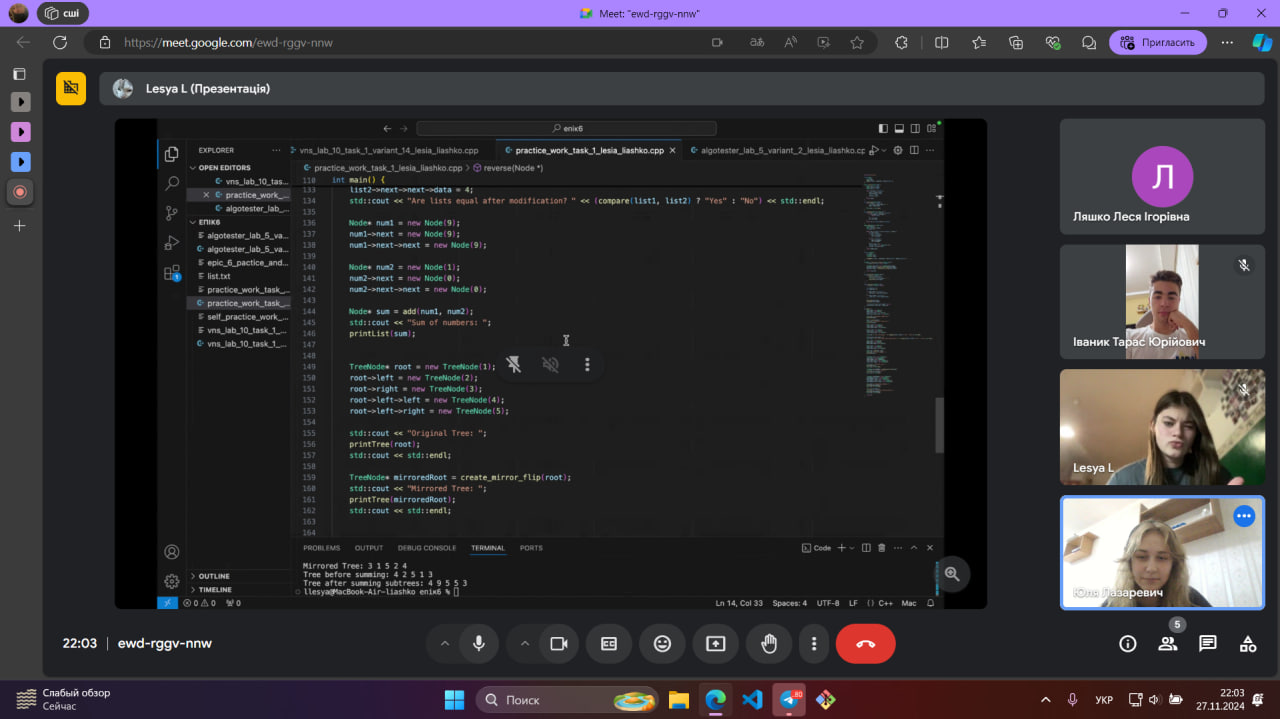
1. **Кооперація з командою:**

* Скрін прогресу по Трелло



Відстежували прогрес всієї команди завдяки дошці Trello

* Скрін з 2-ї зустрічі по обговоренню задач Епіку та Скрін прогресу по Трелло



Зустрічалися багато разів для обговорення та спільного виконання епіків.

**Висновки:** Виконуючи цей епік я ознайомилась з динамічними структурами (Чергою, Стеком, Списком, Деревом) та алгоритмом обробки динамічних структур. Також, я написала шість кодів, завдяки яким я закріпила мої знання.