Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту

A blue and white logo

Description automatically generated

**Звіт**

**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:

ВНС Лабораторної Роботи № 10

Алготестер Лабораторної Роботи № 5

Алготестер Лабораторної Роботи № 7-8

Практичних Робіт до блоку № 6

**Виконав(ла):**

Студент групи ШІ-11

Зубрицький Арсеній Юрійович

# **Тема роботи:**

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

# **Мета роботи:**

Реалізувати різні динамічні структури і функції для роботи з ними.

# **Теоретичні відомості:**

Теоретичні відомості з переліком важливих тем:

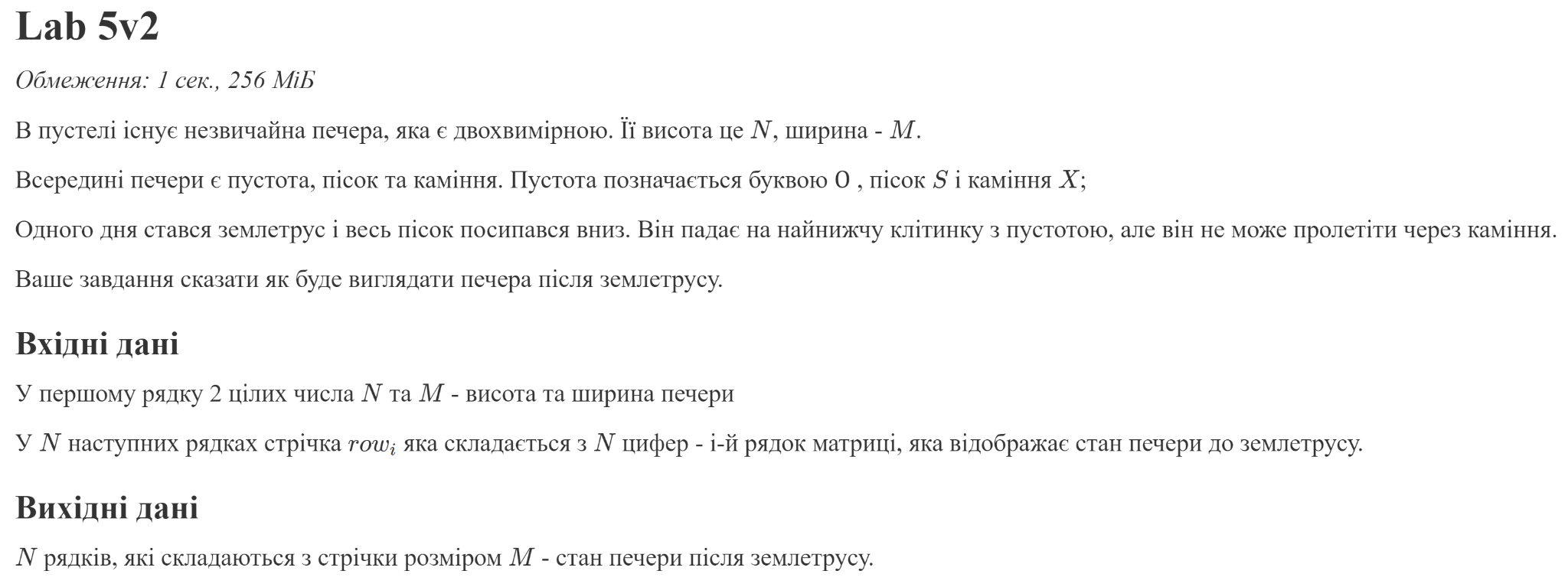
1. Основи Динамічних Структур Даних:
   * Вступ до динамічних структур даних: визначення та важливість
   * Виділення пам'яті для структур даних (stack і heap)
   * Приклади простих динамічних структур: динамічний масив
2. Стек:
   * Визначення та властивості стеку
   * Операції push, pop, top: реалізація та використання
   * Приклади використання стеку: обернений польський запис, перевірка балансу дужок
   * Переповнення стеку
3. Черга:
   * Визначення та властивості черги
   * Операції enqueue, dequeue, front: реалізація та застосування
   * Приклади використання черги: обробка подій, алгоритми планування
   * Розширення функціоналу черги: пріоритетні черги
4. Зв'язні Списки:
   * Визначення однозв'язного та двозв'язного списку
   * Принципи створення нових вузлів, вставка між існуючими, видалення, створення кільця(circular linked list)
   * Основні операції: обхід списку, пошук, доступ до елементів, об'єднання списків
   * Приклади використання списків: управління пам'яттю, FIFO та LIFO структури
5. Дерева:
   * Вступ до структури даних "дерево": визначення, типи
   * Бінарні дерева: вставка, пошук, видалення
   * Обхід дерева: в глибину (preorder, inorder, postorder), в ширину
   * Застосування дерев: дерева рішень, хеш-таблиці
   * Складніші приклади дерев: AVL, Червоно-чорне дерево
6. Алгоритми Обробки Динамічних Структур:
   * Основи алгоритмічних патернів: ітеративні, рекурсивні
   * Алгоритми пошуку, сортування даних, додавання та видалення елементів

* Індивідуальний план опрацювання теорії:
  + [C++ ⦁ Теорія ⦁ Урок 139 ⦁ ADT ⦁ Однозв'язний список](https://www.youtube.com/watch?v=-25REjF_atI)
  + [C++ ⦁ Теорія ⦁ Урок 140 ⦁ ADT ⦁ Двозв'язний список](https://www.youtube.com/watch?v=QLzu2-_QFoE)

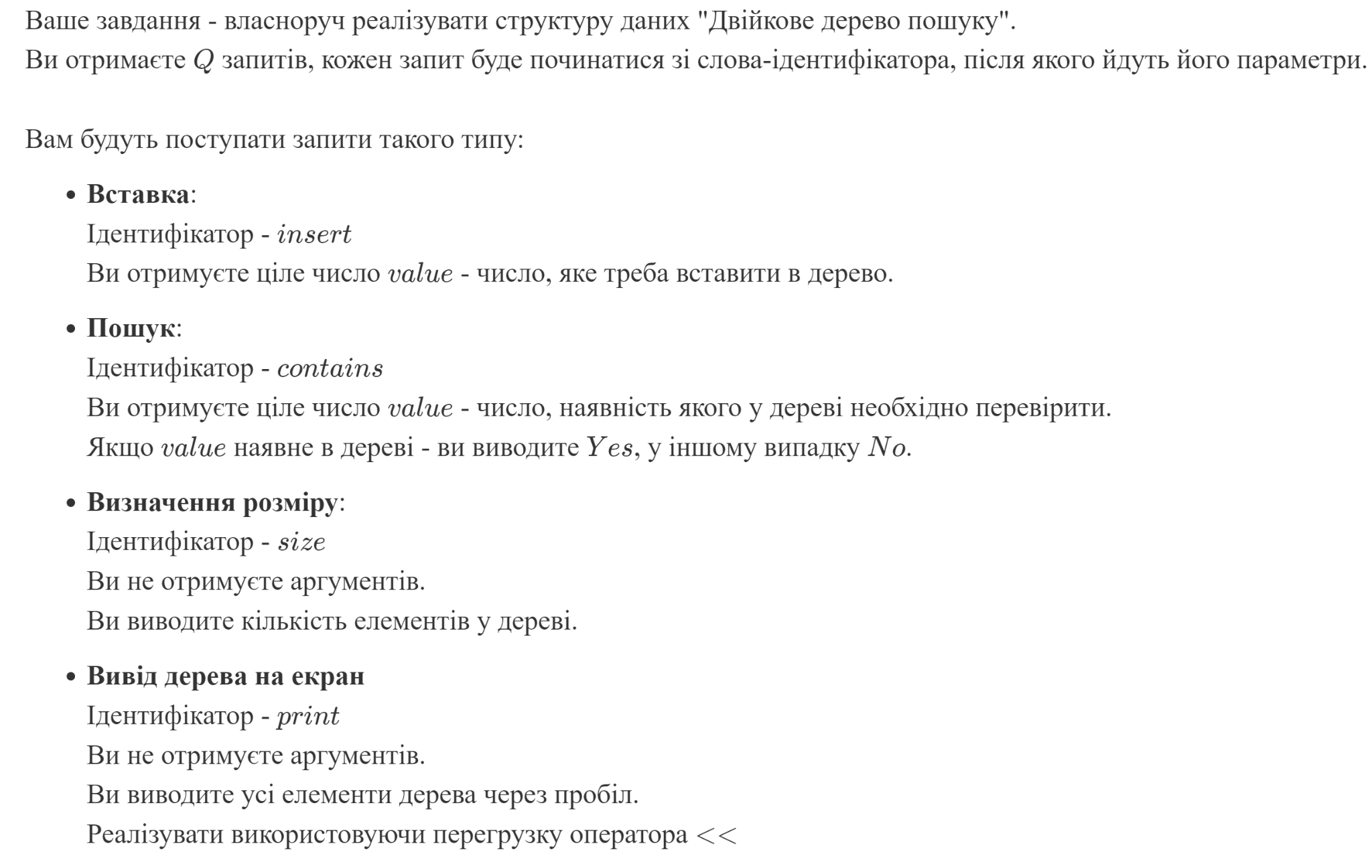
# **Виконання роботи:**

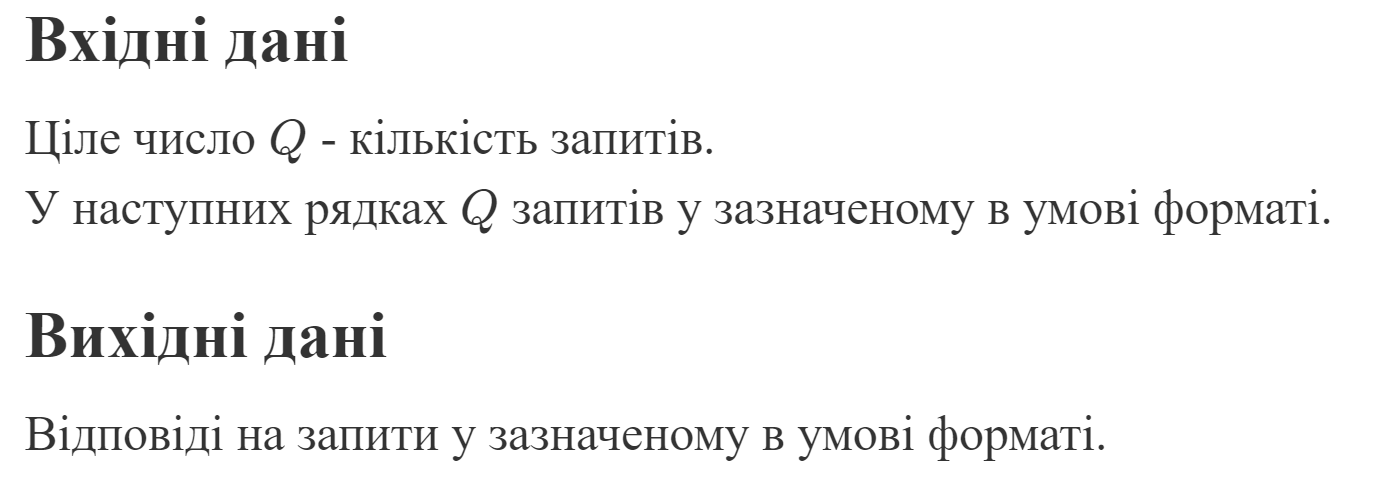
## **1. Опрацювання завдання та вимог до програм та середовища:**

Завдання №1 **Vns Lab\_5\_variant\_2**

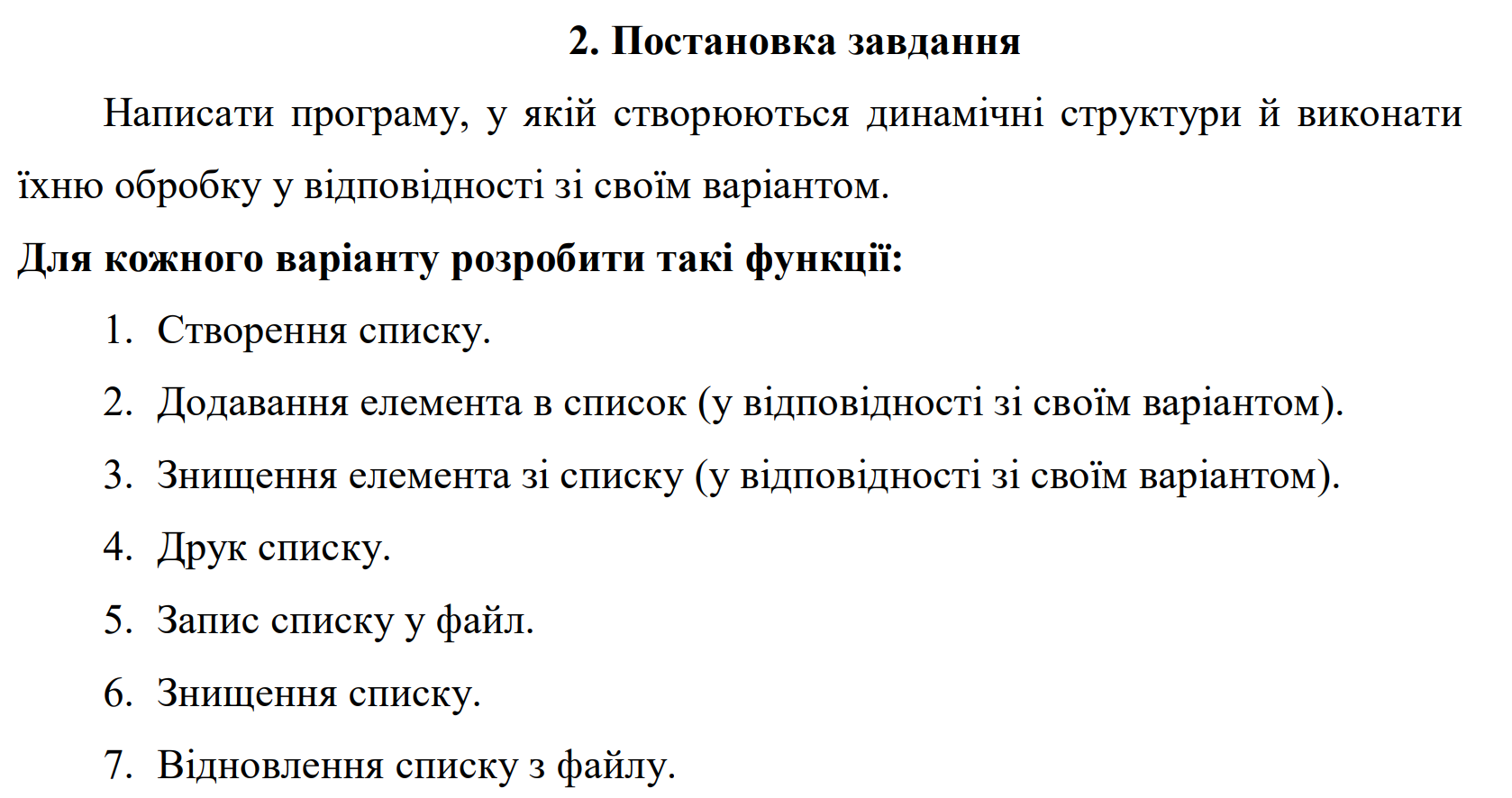


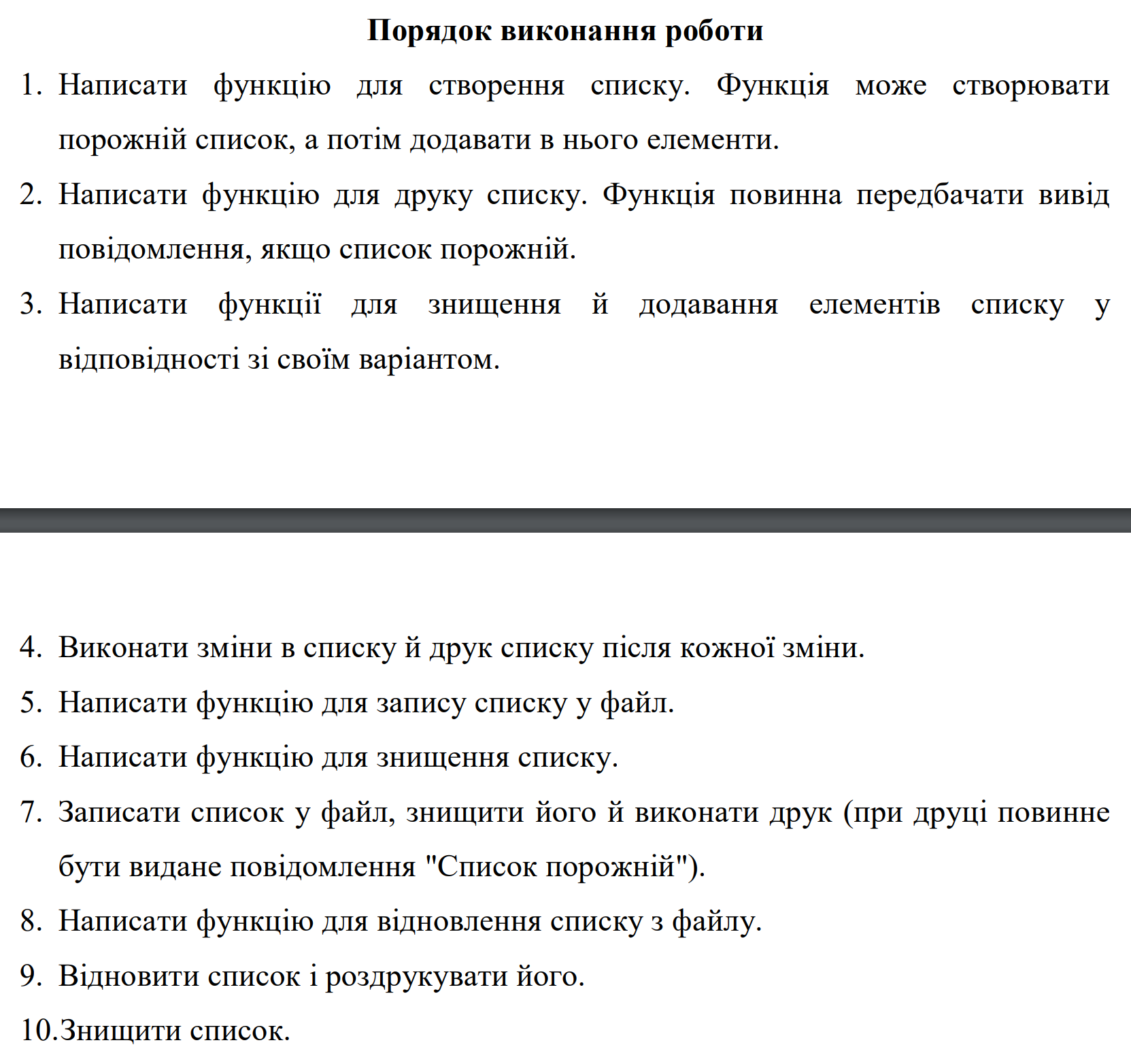
Завдання №2 **VNS Lab\_7\_8\_variant\_3**

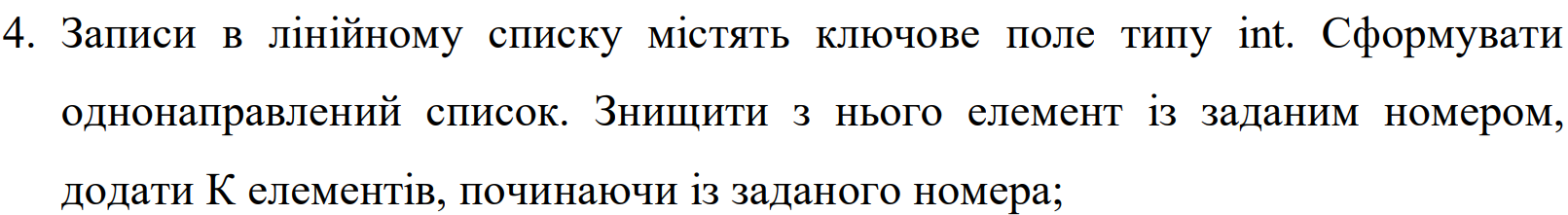




Завдання №3 **VNS Lab\_10\_task\_1\_4**

****

****

****

Завдання №4 **Робота з різними типами структурами**

# **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

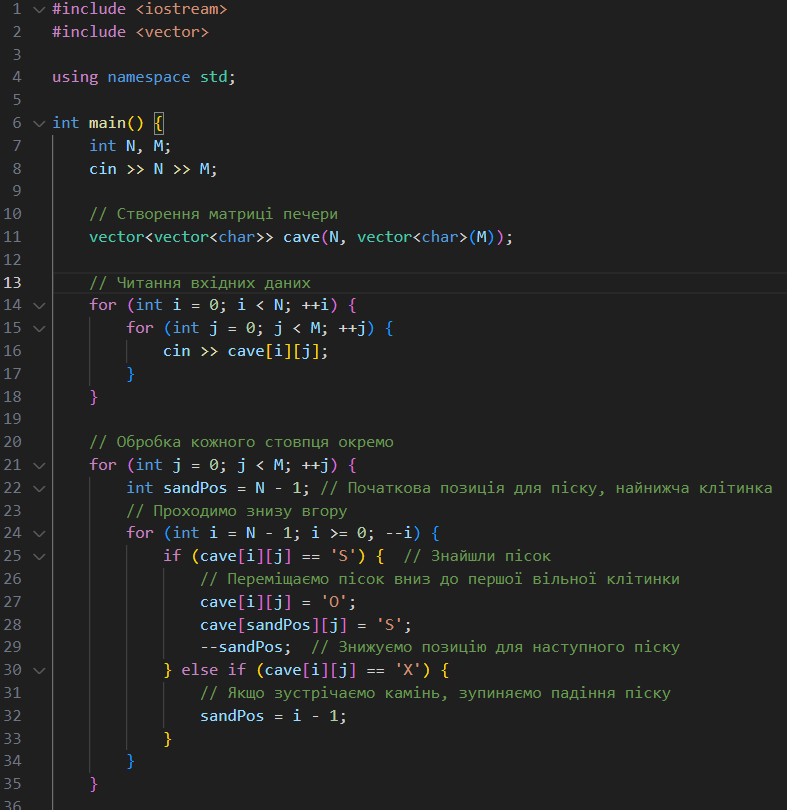
### Мета задачі

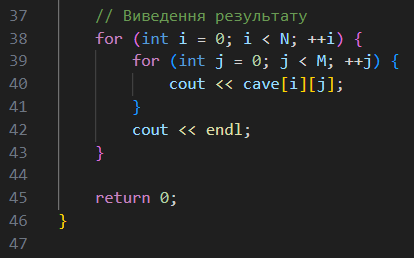
***Розуміння структур даних:*** Реалізація методу підрахунку сум підвузлів бінарного дерева покращує розуміння структури бінарного дерева. Це один з багатьох методів роботи з бінарними деревами.

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

## **2. Код програм**

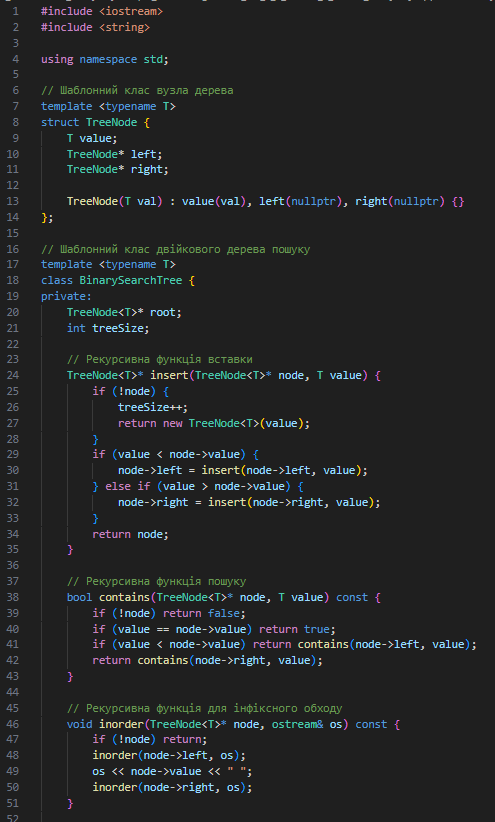
Завдання №1



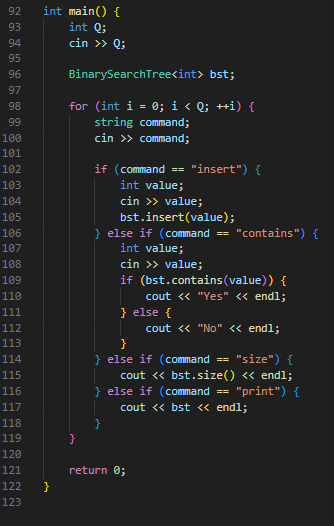


Посилання на pull-request: <https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/856/files#diff-d646b2569ac0ec70707b5c30bfd9d0cc49ddea95b31b0798aa6f215458ed7251>

Завдання №2

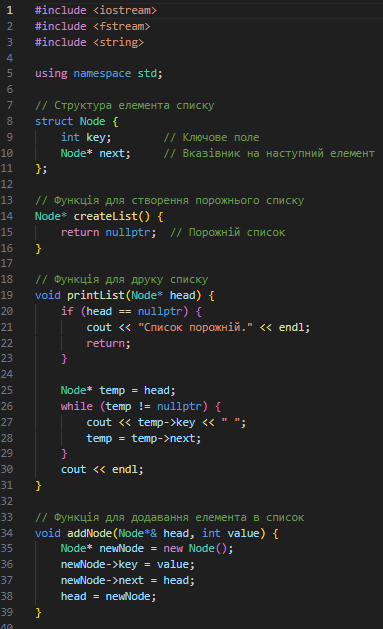


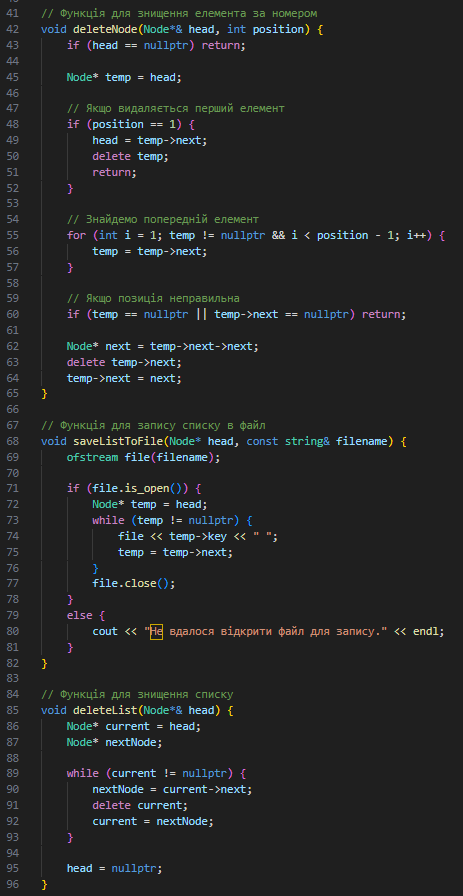


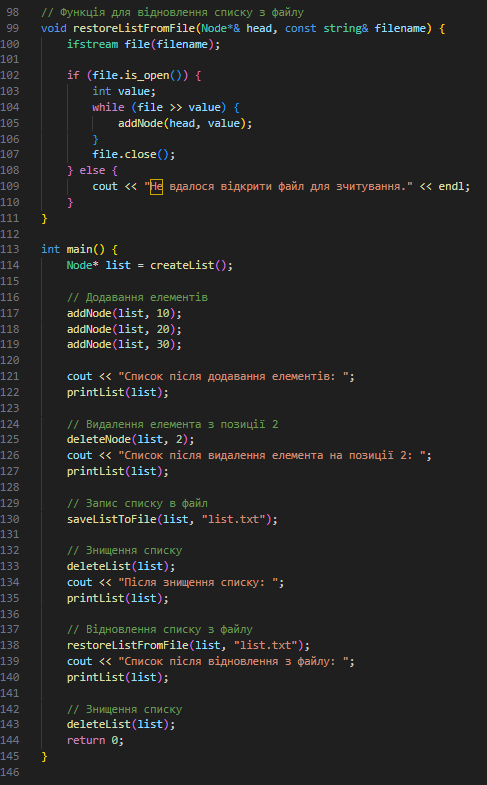


Посилання на pull-request: <https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/856/files#diff-b27b1a7faac4b91b631b06b258e23844e88ca83553f75b4bda5bf60b6ddcb49a>

Завдання №3

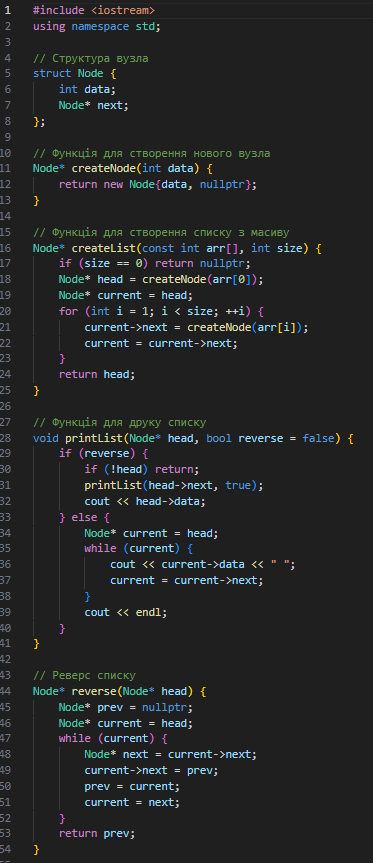


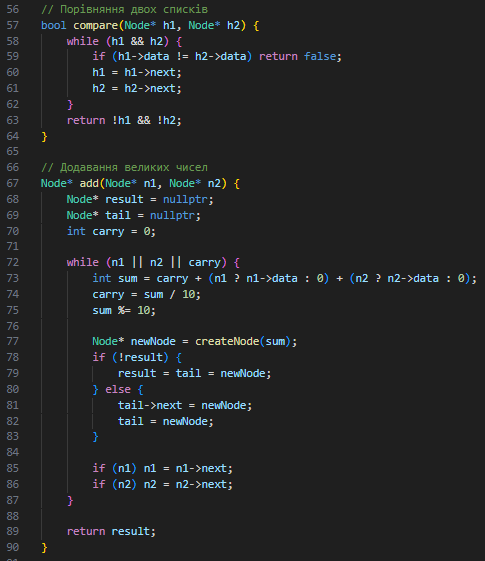


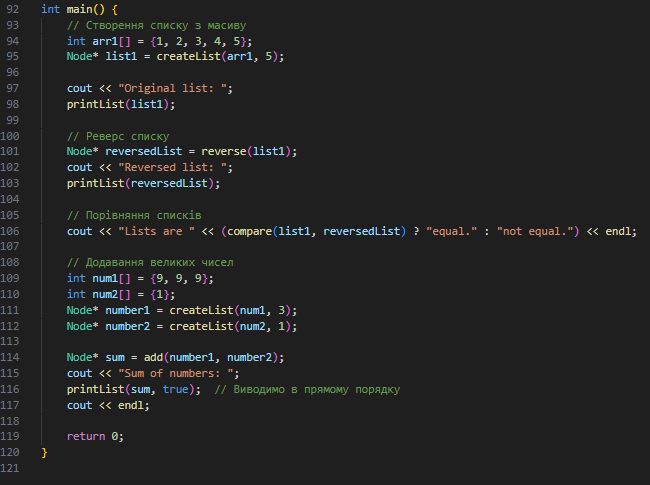


Посилання на pull-request: <https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/856/files#diff-5d94f28e7eac5e9a642af83e4efc23e98b1736bc84c4ebfe6e2755af4671dab9>

Завдання №4\_1



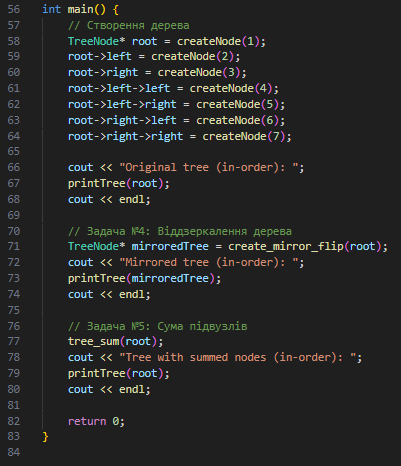




Посилання на pull-request: <https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/856/files#diff-ac5d1f8cf43bcf774b9be07707349d19a1746ebc64411040a81cb2c5d05e31db>

Завдання №4\_2





Посилання на pull-request: <https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/856/files#diff-a4a8f761d55cf3d8e22a9f905c9e52ac80e08022996dad92a83bdc084d2445c0>

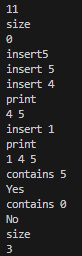
## **4. Результати виконання завдань, тестування та фактично затрачений час:**

Завдання №1



Час затрачений на виконання завдання: 1 год

Завдання №2



Час затрачений на виконання завдання: 4 год

Завдання №3





Час затрачений на виконання завдання: 2 год

Завдання №4\_1



Час затрачений на виконання завдання: 3 год

Завдання №4\_1



Час затрачений на виконання завдання: 3 год

# **Висновки:** В ході виконання робіт з epic\_6 я опрацював динамічні структури данних такі як черга, стек, зв’язні списки, дерева. На практиці реалізував основні принципи роботи з ними обхід, пошук додавання елементів та видалення.