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.
- 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
- 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 рядка 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 R_i, C_i - 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 tan - 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) 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** і-го елементу
  Ідентифікатор - get
  Ви отримуєте ціле число - index, індекс елемента.
  Ви виводите значення елемента за індексом.
- **Модифікація значення** іі-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. $379 \implies 9 \rightarrow 7 \rightarrow 3$);
- 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.

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

Завдання №1 – VNS Lab 10 – Variant 10.

vns_lab_10_task_10_yuliia_lazarevych.cpp > ...

```cpp
#include <iostream>
#include <fstream>
using namespace std;

//вузол двонаправленого списку
struct Node {
    int value;
    Node* next;
    Node* prev;

    Node(int val) : value(val), next(nullptr), prev(nullptr) {}
};

class DoublyLinkedList {
private:
    Node* head;
    Node* tail;

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

    ~DoublyLinkedList() {
        destroy_list(); //знищення списку
    }

    bool is_empty() const {
        return head == nullptr;
    }

    // створення списку
    void create_list() {
        head = tail = nullptr;
        cout << "Список створено.\n";
    }

    //друк списку
    void print_list() const {
        if (is_empty()) {
            cout << "Список порожній.\n";
            return;
        }
        Node* current = head;
        while (current) {
            cout << current->value << " <-> ";
            current = current->next;
        }
        cout << "NULL\n";
    }

    //додавання елемента у список
    void add_to_position(int value, int position) {
        Node* new_node = new Node(value);
        if (is_empty() || position <= 0) {
            //додаємо на початок
            new_node->next = head;
```

vns_lab_10_task_10_yuliia_lazarevych.cpp > ...

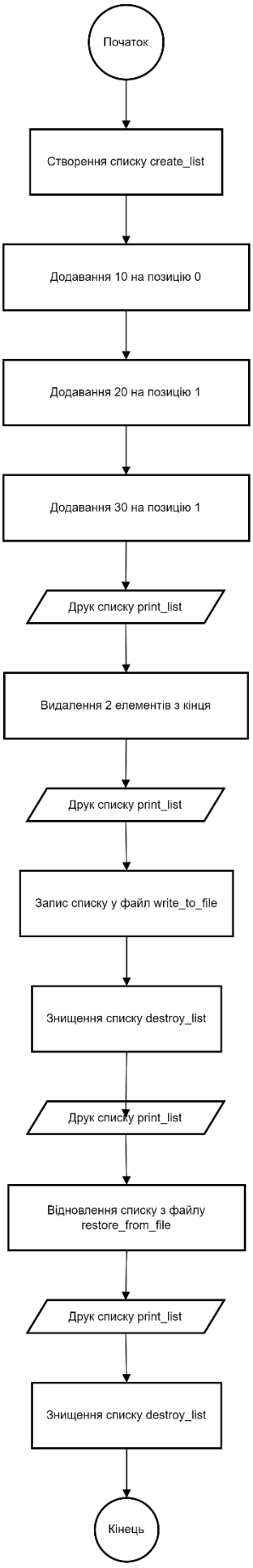
```cpp
class DoublyLinkedList {
    void add_to_position(int value, int position) {
        if (is_empty() || position <= 0) {
            if (head) head->prev = new_node;
            head = new_node;
            if (!tail) tail = head;
        } else {
            Node* current = head;
            int index = 0;
            while (current->next && index < position - 1) {
                current = current->next;
                index++;
            }
            new_node->next = current->next;
            new_node->prev = current;
            if (current->next) current->next->prev = new_node;
            current->next = new_node;
            if (!new_node->next) tail = new_node;
        }
        cout << "Елемент " << value << " додано на позицію " << positio
    }

    //видалення елементів зі списку
    void delete_from_end(int k) {
        while (k > 0 && tail) {
            Node* temp = tail;
            tail = tail->prev;
            if (tail) tail->next = nullptr;
            else head = nullptr;
            delete temp;
            k--;
        }
        cout << k << " елементів видалено з кінця списку.\n";
    }

    //запис списку у файл
    void write_to_file(const string& filename) {
        ofstream file(filename);
        if (!file) {
            cerr << "Помилка відкриття файлу.\n";
            return;
        }
        Node* current = head;
        while (current) {
            file << current->value << " ";
            current = current->next;
        }
        file.close();
        cout << "Список записано у файл " << filename << ".\n";
    }

    //знищення списку
    void destroy_list() {
        while (head) {
            Node* temp = head;
```

vns_lab_10_task_10_yuliia_lazarevych.cpp > ...

```cpp
class DoublyLinkedList {
    void destroy_list() {
        while (head) {
            head = head->next;
            delete temp;
        }
        tail = nullptr;
        cout << "Список знищено.\n";
    }

    //відновлення списку з файлу
    void restore_from_file(const string& filename) {
        destroy_list(); //знищення поточного списку
        ifstream file(filename);
        if (!file) {
            cerr << "Помилка відкриття файлу.\n";
            return;
        }
        int value;
        while (file >> value) {
            add_to_position(value, -1); //додаємо в кінець
        }
        file.close();
        cout << "Список відновлено з файлу " << filename << ".\n";
    }
};

int main() {
    DoublyLinkedList list;

    //створення списку
    list.create_list();

    //додавання елемента у список
    list.add_to_position(10, 0);
    list.add_to_position(20, 1);
    list.add_to_position(30, 1);

    //друк списку
    list.print_list();

    //видалення елементів зі списку
    list.delete_from_end(2);
    list.print_list();

    //запис списку у файл
    list.write_to_file("list.txt");

    //знищення списку
    list.destroy_list();
    list.print_list();

    //відновлення списку з файлу
    list.restore_from_file("list.txt");
    list.print_list();
```

```cpp

    //знищення списку
    list.destroy_list();

    return 0;
}
```

```
Список створено.
Елемент 10 додано на позицію 0.
Елемент 20 додано на позицію 1.
Елемент 30 додано на позицію 1.
10 <-> 30 <-> 20 <-> NULL
0 елементів видалено з кінця списку.
10 <-> NULL
Список записано у файл list.txt.
Список знищено.
Список порожній.
Список знищено.
Елемент 10 додано на позицію -1.
Список відновлено з файлу list.txt.
10 <-> NULL
Список знищено.
Список знищено.
```


Початок
Створення списку create_list
Додавання 10 на позицію 0
Додавання 20 на позицію 1
Додавання 30 на позицію 1
Друк списку print_list
Видалення 2 елементів з кінця
Друк списку print_list
Запис списку у файл write_to_file
Знищення списку destroy_list
Друк списку print_list
Відновлення списку з файлу
restore_from_file
Друк списку print_list
Знищення списку destroy_list
Кінець

Завдання №2 – Algotester Lab 5 – Variant 1.

algotester_lab_5_task_1_yuliia_lazarevych.cpp > ...

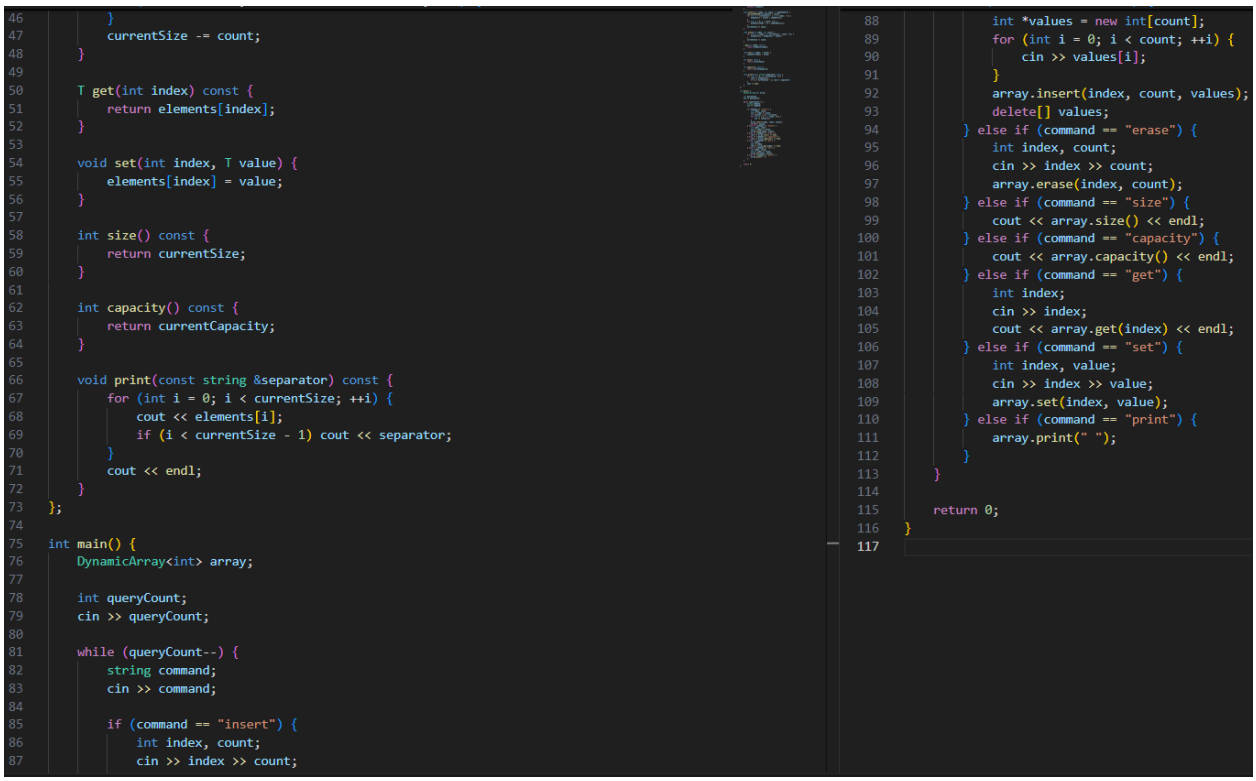
```cpp
#include <iostream>
using namespace std;

int main() {
    uint64_t a; //поточний стан дошки
    int N; //к-ть заклинань
    cin >> a >> N;

    for (int i = 0; i < N; i++) {
        int R, C;
        cin >> R >> C;
        R--; C--; //переходимо до 0-індексації

        //інвертуємо рядок R
        for (int j = 0; j < 8; j++) {
            a ^= (1ULL << (R * 8 + j));
        }

        //інвертуємо стовпець C
        for (int j = 0; j < 8; j++) {
            if (j != R) { //уникаємо подвійної інверсії
                a ^= (1ULL << (j * 8 + C));
            }
        }
    }

    cout << a << endl; //виводимо кінцевий стан
    return 0;
}
```

Створено	Компілятор	Результат	Час (сек.)	Пам'ять (МіБ)	Дії
декілька секунд тому	C++ 23	Зараховано	0.003	1.430	Перегляд

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

algotester_lab_7_8_variant_2_task_1_yuliia_lazarevych.cpp > ...

```cpp
#include <iostream>
using namespace std;

template <typename T>
class DynamicArray {
private:
    T *elements;
    int currentSize;
    int currentCapacity;

    void resizeIfNeeded(int newSize) {
        while (newSize >= currentCapacity) {
            currentCapacity *= 2;
        }
        T *newArray = new T[currentCapacity];
        for (int i = 0; i < currentSize; ++i) {
            newArray[i] = elements[i];
        }
        delete[] elements;
        elements = newArray;
    }

public:
    DynamicArray() : currentSize(0), currentCapacity(1) {
        elements = new T[currentCapacity];
    }

    ~DynamicArray() {
        delete[] elements;
    }

    void insert(int index, int count, T *newElements) {
        resizeIfNeeded(currentSize + count);
        for (int i = currentSize - 1; i >= index; --i) {
            elements[i + count] = elements[i];
        }
        for (int i = 0; i < count; ++i) {
            elements[index + i] = newElements[i];
        }
        currentSize += count;
    }

    void erase(int index, int count) {
        for (int i = index; i < currentSize - count; ++i) {
            elements[i] = elements[i + count];
```



```
        }
        currentSize -= count;
    }

    T get(int index) const {
        return elements[index];
    }

    void set(int index, T value) {
        elements[index] = value;
    }

    int size() const {
        return currentSize;
    }

    int capacity() const {
        return currentCapacity;
    }

    void print(const string &separator) const {
        for (int i = 0; i < currentSize; ++i) {
            cout << elements[i];
            if (i < currentSize - 1) cout << separator;
        }
        cout << endl;
    }
};

int main() {
    DynamicArray<int> array;

    int queryCount;
    cin >> queryCount;

    while (queryCount--) {
        string command;
        cin >> command;

        if (command == "insert") {
            int index, count;
            cin >> index >> count;
            int *values = new int[count];
            for (int i = 0; i < count; ++i) {
                cin >> values[i];
            }
            array.insert(index, count, values);
            delete[] values;
        } else if (command == "erase") {
            int index, count;
            cin >> index >> count;
            array.erase(index, count);
        } else if (command == "size") {
            cout << array.size() << endl;
        } else if (command == "capacity") {
            cout << array.capacity() << endl;
        } else if (command == "get") {
            int index;
            cin >> index;
            cout << array.get(index) << endl;
        } else if (command == "set") {
            int index, value;
            cin >> index >> value;
            array.set(index, value);
        } else if (command == "print") {
            array.print(" ");
        }
    }

    return 0;
}
```

Створено	Компілятор	Результат	Час (сек.)	Пам'ять (МіБ)	Дії
декілька секунд тому	C++ 23	Зараховано	0.006	1.371	Перегляд

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

algotester_lab_7_8_variant_1_task_1_yuliia_lazarevych.cpp > ...

```cpp
#include <iostream>
using namespace std;

template <typename T> //використовую шаблонний клас для роботи з будь-яким типом (в цьому випадку int)
class DoublyLinkedList {
private:
    struct Node { //власна структура для вузла двозв'язного списку
        T data;       //дані, що зберігаються в вузлі (тип T, який задається під час створення списку)
        Node* prev;   //вказівник на попередній вузол
        Node* next;   //вказівник на наступний вузол
        Node(T val) : data(val), prev(nullptr), next(nullptr) {} //конструктор для ініціалізації вузла
    };

    Node* head; //початок списку
    Node* tail; //кінець списку
    size_t listSize; //розмір списку

    //використовую допоміжну функцію для доступу до вузла за індексом (позначаю const, оскільки вона не змінює стан списку)
    Node* getNodeAt(int index) const {
        Node* current = head;
        while (index-- && current) { //проходимо по списку до індексу
            current = current->next;
        }
        return current;
    }

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr), listSize(0) {} //конструктор за замовчуванням

    ~DoublyLinkedList() { //деструктор для звільнення пам'яті
        Node* current = head;
        while (current) {
            Node* toDelete = current;
            current = current->next;
            delete toDelete; //видалення всіх вузлів списку
        }
    }

    //вставка елементів на задану позицію в список
    void insert(int index, const T* elements, size_t n) {
        if (index < 0 || index > listSize) return; //перевірка на коректність індексу

        Node* prevNode = nullptr;
        if (index > 0) prevNode = getNodeAt(index - 1); //знаходимо попередній вузол, якщо індекс більше 0
        Node* nextNode = prevNode ? prevNode->next : head; //наступний вузол
```

```
46
47          for (size_t i = 0; i < n; ++i) {
48              Node* newNode = new Node(elements[i]); //створюємо новий вузол
49              if (prevNode) prevNode->next = newNode; //зв'язуємо попередній вузол з новим
50              newNode->prev = prevNode; //зв'язуємо новий вузол з попереднім
51
52              if (nextNode) nextNode->prev = newNode; //зв'язуємо новий вузол з наступним
53              newNode->next = nextNode; //зв'язуємо новий вузол з наступним
54
55              prevNode = newNode;
56              if (index == 0 && i == 0) head = newNode; //якщо індекс 0, то новий вузол стає головою списку
57              if (index == listSize) tail = newNode; //якщо індекс на кінці, новий вузол стає хвостом списку
58          }
59
60          if (prevNode && !nextNode) tail = prevNode; //якщо наступного вузла немає, то змінюємо хвіст
61          listSize += n; //оновлюємо розмір списку
62      }
63
64      //видалення елементів із списку, починаючи з певного індексу
65      void erase(int index, int n) {
66          if (index < 0 || index >= listSize || n <= 0) return; //перевірка коректності індексів
67
68          Node* startNode = getNodeAt(index); //знаходимо вузол, з якого почнемо видаляти
69          Node* endNode = startNode;
70          for (int i = 0; i < n && endNode; ++i) endNode = endNode->next; //знаходимо вузол, де потрібно завершити видалення
71
72          Node* prevNode = startNode->prev; //попередній вузол для початкового
73          Node* nextNode = endNode; //наступний вузол для кінцевого
74
75          if (prevNode) prevNode->next = nextNode; //зв'язуємо попередній вузол з наступним
76          if (nextNode) nextNode->prev = prevNode; //зв'язуємо наступний вузол з попереднім
77
78          if (!prevNode) head = nextNode; //якщо видаляється перший елемент, то змінюємо голову
79          if (!nextNode) tail = prevNode; //якщо видаляється останній елемент, то змінюємо хвіст
80
81          while (startNode != endNode) { //видаляємо вузли по черзі
82              Node* toDelete = startNode;
83              startNode = startNode->next;
84              delete toDelete;
85          }
86
87          listSize -= n; //оновлюємо розмір списку
88      }
```

```
    //функція для отримання розміру списку
    size_t size() const {
        return listSize;
    }

    //функція для отримання значення елемента за індексом
    T get(int index) const {
        Node* node = getNodeAt(index);
        return node ? node->data : T(); //повертаємо значення, якщо вузол існує
    }

    //функція для зміни значення елемента за індексом
    void set(int index, T value) {
        Node* node = getNodeAt(index);
        if (node) node->data = value; //змінюємо значення вузла
    }

    //перевантаження оператора << для виведення списку
    friend ostream& operator<<(ostream& os, const DoublyLinkedList& list) {
        Node* current = list.head;
        while (current) {
            os << current->data; //виводимо значення вузла
            if (current->next) os << " "; //додаємо пробіл між елементами
            current = current->next;
        }
        return os;
    }
};

int main() {
    DoublyLinkedList<int> list; //використовуємо шаблонний клас для int
    int Q;
    cin >> Q;

    while (Q--) {
        string command;
        cin >> command;

        if (command == "insert") {
            int index, N;
            cin >> index >> N;
            int* elements = new int[N];
            for (int i = 0; i < N; ++i) cin >> elements[i];
            list.insert(index, elements, N);
```

```
            delete[] elements;
        } else if (command == "erase") {
            int index, n;
            cin >> index >> n;
            list.erase(index, n);
        } else if (command == "size") {
            cout << list.size() << endl;
        } else if (command == "get") {
            int index;
            cin >> index;
            cout << list.get(index) << endl;
        } else if (command == "set") {
            int index, value;
            cin >> index >> value;
            list.set(index, value);
        } else if (command == "print") {
            cout << list << endl;
        }
    }

    return 0;
}
```

Створено	Компілятор	Результат	Час (сек.)	Пам'ять (МіБ)	Дії
декілька секунд тому	C++ 23	Зараховано	0.008	1.430	Перегляд

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

practice_work_team_tasks_yuliia_lazarevych.cpp > ...

```cpp
#include <iostream>
#include <vector>
using namespace std;

//структура вузла списку
struct Node {
    int data;
    Node* next;
    Node(int value) : data(value), next(nullptr) {}
};

//вставка вузлів у список
Node* insert(Node* head, const vector<int>& values) {
    Node* current = nullptr;
    for (int value : values) {
        Node* newNode = new Node(value);
        if (!head) {
            head = newNode;
            current = head;
        } else {
            current->next = newNode;
            current = newNode;
        }
    }
    return head;
}

//виведення списку
void printLL(Node* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

//реверс списку
Node* reverse(Node* head) {
    Node* prev = nullptr;
    Node* current = head;
    while (current) {
        Node* next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}

//порівняння списків
bool compare(Node* head1, Node* head2) {
    while (head1 && head2) {
        if (head1->data != head2->data) return false;
        head1 = head1->next;
        head2 = head2->next;
```

practice_work_team_tasks_yuliia_lazarevych.cpp > ...

```cpp
bool compare(Node* head1, Node* head2) {
    while (head1 && head2) {
    }
    return head1 == nullptr && head2 == nullptr;
}

//додавання великих чисел
Node* add(Node* head1, Node* head2) {
    Node* dummy = new Node(0);
    Node* current = dummy;
    int carry = 0;
    while (head1 || head2 || carry) {
        int sum = carry + (head1 ? head1->data : 0) + (head2 ? head2->data : 0);
        carry = sum / 10;
        current->next = new Node(sum % 10);
        current = current->next;
        if (head1) head1 = head1->next;
        if (head2) head2 = head2->next;
    }
    return dummy->next;
}

//структура вузла дерева
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int value) : data(value), left(nullptr), right(nullptr) {}
};

//віддзеркалення дерева
TreeNode* createMirrorFlip(TreeNode* root) {
    if (!root) return nullptr;
    TreeNode* newNode = new TreeNode(root->data);
    newNode->left = createMirrorFlip(root->right);
    newNode->right = createMirrorFlip(root->left);
    return newNode;
}

//обхід і виведення дерева
void printTree(TreeNode* root) {
    if (!root) return;
    printTree(root->left);
    cout << root->data << " ";
    printTree(root->right);
}

//запис суми підвузлів
int treeSum(TreeNode* root) {
    if (!root) return 0;
    int leftSum = treeSum(root->left);
    int rightSum = treeSum(root->right);
    if (root->left || root->right) root->data = leftSum + rightSum;
    return root->data;
}
```

practice_work_team_tasks_yuliia_lazarevych.cpp > ...

```cpp
int treeSum(TreeNode* root) {

int main() {
    //завдання 1: реверс списку
    vector<int> values = {1, 2, 3, 4, 5};
    Node* list = insert(nullptr, values);
    cout << "Original list: ";
    printLL(list);
    Node* reversed = reverse(list);
    cout << "Reversed list: ";
    printLL(reversed);

    //завдання 2: порівняння списків
    cout << "Lists are " << (compare(list, reversed) ? "identical" : "differe

    //завдання 3: додавання великих чисел
    Node* num1 = insert(nullptr, {9, 9, 9});
    Node* num2 = insert(nullptr, {1});
    Node* sum = add(num1, num2);
    cout << "Sum of numbers: ";
    printLL(sum);

    //завдання 4: віддзеркалення дерева
    TreeNode* root = new TreeNode(10);
    root->left = new TreeNode(5);
    root->right = new TreeNode(15);
    cout << "Original tree: ";
    printTree(root);
    cout << endl;
    TreeNode* mirrored = createMirrorFlip(root);
    cout << "Mirrored tree: ";
    printTree(mirrored);
    cout << endl;

    //завдання 5: сума підвузлів
    cout << "Tree before sum update: ";
    printTree(root);
    cout << endl;
    treeSum(root);
    cout << "Tree after sum update: ";
    printTree(root);
    cout << endl;

    return 0;
}
```

```
Original list: 1 2 3 4 5
Reversed list: 5 4 3 2 1
Lists are different
Sum of numbers: 0 0 0 1
Original tree: 5 10 15
Mirrored tree: 15 10 5
Tree before sum update: 5 10 15
Tree after sum update: 5 20 15
```

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

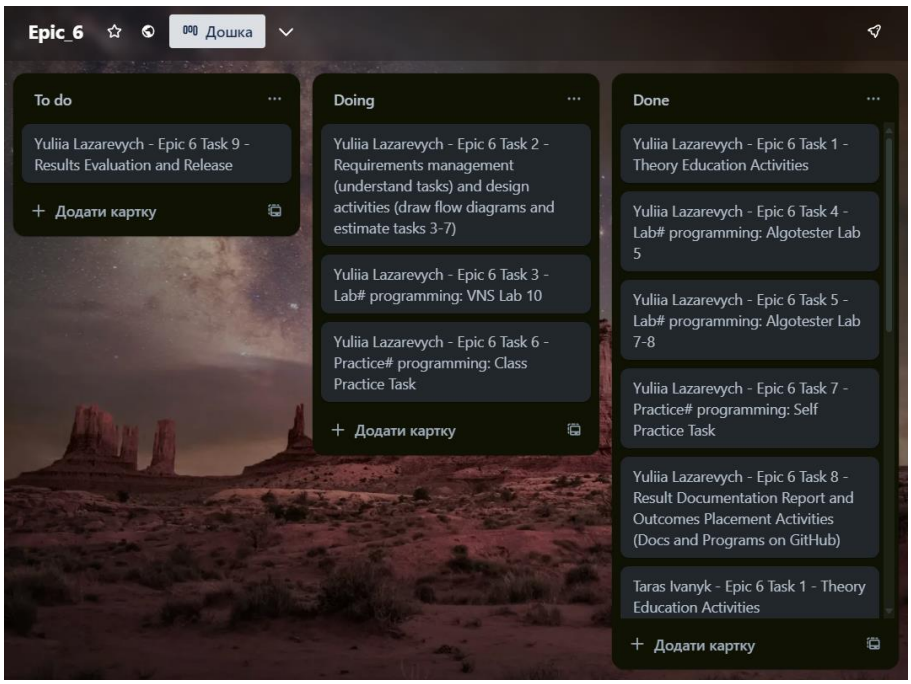
practice_work_self_algotester_tasks_yuliia_lazarevych.cpp > ...

```cpp
//лаба 5 варіант 3(гори)

#include <iostream>
#include <vector>
using namespace std;

int main() {
    int N, M, x, y;
    cin >> N >> M;
    cin >> x >> y;

    //зменшуємо координати до 0-індексації
    x--;
    y--;

    //ініціалізуємо матриці висот
    vector<vector<int>> matrix(N, vector<int>(M, 0));

    //обчислюємо максимальної висоти
    int maxHeight = max(x, N - x - 1) + max(y, M - y - 1);

    //заповнюємо матрицю висот
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            int distance = abs(x - i) + abs(y - j);
            matrix[i][j] = maxHeight - distance;
        }
    }

    //виводимо матриці висот
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            cout << matrix[i][j] << (j == M - 1 ? '\n' : ' ');
        }
    }

    return 0;
}
```

Створено	Компілятор	Результат	Час (сек.)	Пам'ять (МіБ)	Дії
хвилину тому	C++ 23	Зараховано	0.076	6.262	Перегляд

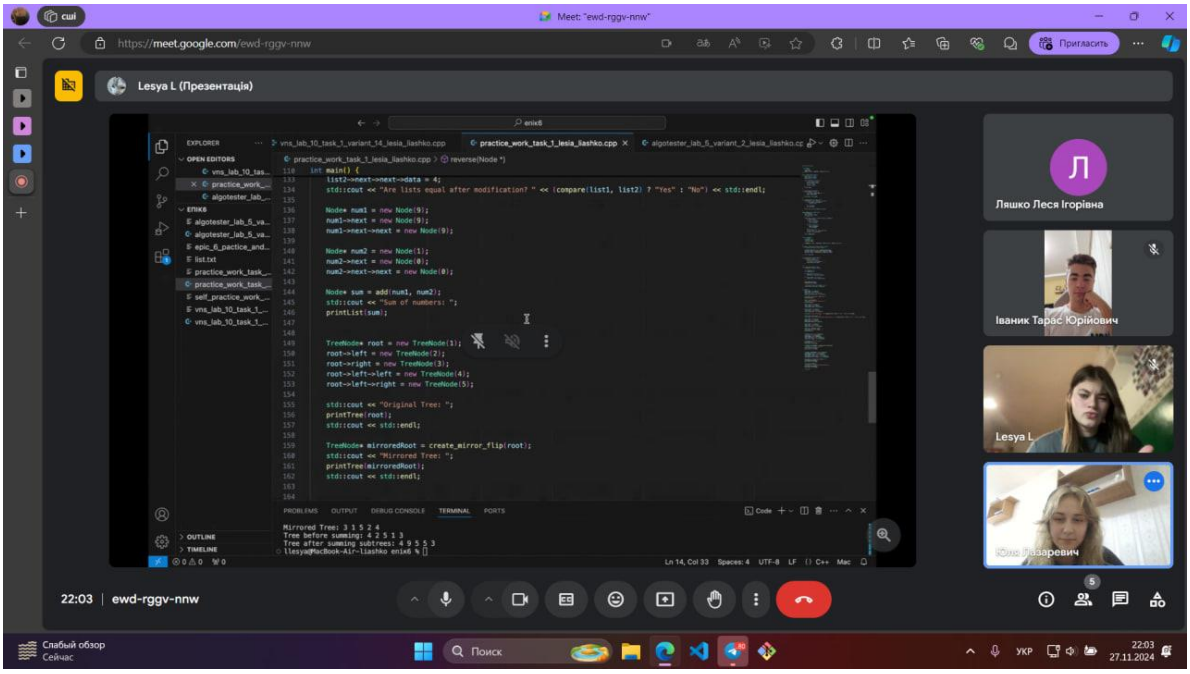
3. Кооперація з командою:

- 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.