МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет електроніки і комп’ютерних технологій

**Звіт**

Про виконання лабораторної роботи № 4

**«Структури даних. Лінійні списки.»**

**Виконав:**

Студент групи ФЕП-14с

Грицюк Максим Олегович

**Перевірив:**

ас. Баран М. О.

Львів 2025

**Мета:** Навчитися обчислювати кількість розміщень, перестановок і сполучень з повтореннями та без повторень, а також генерувати їх у лексикографічному порядку.

**Обладнання:**

Комп'ютер з встановленим програмним забезпеченням(Intellij IDEA Ultimate).

**Теоретичні відомості**

**Поняття про динамічну пам’ять та вказівники.**

Глобальні змінні зберігаються у ділянці пам’яті, яка називається

сегментом даних. Локальні змінні, що використовуються у функціях та існують тільки протягом їх роботи зберігаються у сегменті стеку. Решта базової пам’яті є динамічною і використовується для збереження динамічних змінних.

Динамічну пам’ять ще часто називають купою (англ. heap – купа).

Доступ до значення динамічної змінної здійснюється за її адресою у пам’яті. Для збереження адрес динамічних змінних використовуються спеціальні статичні змінні посилального типу – вказівники (англ. Pointer). Значенням вказівника є адреса в області динамічної пам’яті, де зберігається певний елемент даних.

Зв’язний лінійний список – це набір однотипних компонентів, які

зв’язані між собою за допомогою вказівників. Зв’язані лінійні списки бувають таких типів:

• однозв’язний лінійний список;

• двозв’язний лінійний список;

• стек;

• черга:

• дек.

**Хід роботи**

**Завдання №1.1 Створити нову бібліотеку з необхідними файлами (файли Comb.java, Lab\_3\_1.java).**

Я створив нову Java-бібліотеку, яка складається з двох файлів: Comb.java - для реалізації логіки обробки даних, та Lab\_3\_1.java - де реалізована основна функція main() та меню для взаємодії з користувачем.

****Завдання №1.2** Оголосити клас з основною структурою списку (Node) та задати інформаційне поле, пов’язане з заданим типом даних.**

У файлі Comb.java я створив клас Node, в якому оголосив поле data типу int, а також посилання на наступний елемент списку (next). Це дозволяє зберігати цілі числа у списку.

****Завдання №1.3** Реалізувати функції додавання, видалення елементів та виводу вмісту списку.**

Я реалізував методи:

push(int value) - для додавання елементів у стек,

pop() - для видалення верхнього елемента стека,

show() - для відображення вмісту стека.

****Завдання №1.4** Створити головний клас з функцією main() та реалізувати меню взаємодії з користувачем.**

У файлі Lab\_3\_1.java реалізовано меню, яке дозволяє: додавати елементи до стеку, видаляти елементи, переглядати вміст.

Меню працює у циклі до завершення програми.

****Завдання №2.1** Доповнити реалізацію функціями для роботи з чергою: enqueue() **та** dequeue()**.****

Я розширив функціонал у файлі Comb.java, додавши методи enqueue(int value) та dequeue(), які реалізують додавання і видалення елементів черги.  
Для реалізації черги я використав зв’язаний список, у якому нові елементи додаються в кінець, а видаляються з початку. Це дозволяє зберігати порядок надходження даних.

****Завдання №2.2** Занести прототипи нових функцій у відповідні класи.**

Всі нові методи було оголошено у відповідному класі. В Java ми не використовуємо окремі заголовні файли, але методи були оголошені у класі Comb, що виконує функцію логіки для роботи з чергою та стеком.

****Завдання №2.3** Додати обробку черги в main()та розширити меню.**

Я модифікував клас Lab\_3\_1, додавши у меню пункти для роботи з чергою: додавання елемента в чергу, видалення елемента з черги, перегляд вмісту черги.

Це дало змогу користувачеві взаємодіяти з чергою так само зручно, як і зі стеком.

****Завдання №2.4** Протестувати програму.**

Після додавання нових функцій я скомпілював і запустив програму. Черга працює коректно: елементи додаються в правильному порядку, а видалення відбувається з початку.

****Завдання №3.1** Реалізувати функції для двозв’язного списку:**

Я реалізував вказані методи у файлі Comb.java, використовуючи двозв’язний список. Для цього створив нову структуру вузла (Node) з додатковим полем previous, яке зберігає посилання на попередній елемент.

Всі функції працюють з урахуванням коректного оновлення обох посилань (next і previous), що забезпечує стабільність двозв’язного списку.

****Завдання №3.2** Оновити структуру** Node **та прототипи методів.**

У класі Node я додав поле previous, щоб забезпечити двозв’язність. Це дозволило переміщатись у списку як вперед, так і назад. Усі нові методи були оголошені у відповідному класі. Структура стала зручнішою для реалізації складніших операцій.

****Завдання №3.3** Створити окремий клас** main() **для роботи з двозв’язним списком.**

Я створив новий Java-файл, у якому реалізував головний метод main(). Меню цього блоку дозволяє виконувати всі дії, реалізовані у попередньому завданні:

додавання в початок/кінець/після елемента,

видалення з початку/кінця/всередині,

пошук елементів,

перегляд поточного стану списку.

Інтерфейс користувача простий, зручний і дозволяє тестувати функціонал на будь-яких даних.

****Завдання №3.4** Компіляція та демонстрація роботи.**

Після завершення реалізації я скомпілював проект та перевірив коректність роботи кожної функції. Під час тестування було використано довільні ключі та значення, які дозволили переконатись у правильності логіки обробки двозв’язного списку. Усі дії відбуваються у правильному порядку, без помилок.

### **Висновки:** ця робота допомогла мені краще зрозуміти принципи роботи динамічних структур даних у мові Java та важливість правильної організації коду. Також, у ході виконання лабораторної роботи я реалізував стек та чергу на основі однозв’язного списку, реалізував повноцінний двозв’язний список з розширеними методами додавання, видалення та пошуку. Навчився будувати логіку роботи зі структурами даних у Java.