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

**Виконала:**

Студентка групи ШІ-11

Андрусишин Соломія Володимирівна

*Тема роботи:* Вивчення основ динамічних структур даних у C++: стек, черга, зв'язні списки та дерева, а також алгоритмів обробки цих структур, включаючи операції додавання, видалення елементів та пошук. Розгляд основних принципів виділення пам'яті, переповнення стеку, обробки подій через чергу, а також обробки дерев різних типів (бінарних, AVL, червоно-чорних дерев) за допомогою ітеративних та рекурсивних алгоритмів.

*Мета роботи:* **:** Ознайомитися з основними динамічними структурами даних у C++, зокрема стеком, чергою, зв'язними списками та деревами, вивчити їх властивості та операції (push, pop, enqueue, dequeue тощо). Опанувати основні алгоритми пошуку, сортування, вставки та видалення елементів у цих структурах. Зрозуміти принципи виділення пам'яті для динамічних структур та їх обробку, включаючи випадки переповнення та особливості обробки складних дерев.

*Теоретичні відомості:*

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 опрацьовування теорії:*

***1.Основи Динамічних Структур Даних:***

* Джерела:
* <https://acode.com.ua/urok-111-stek-i-kupa/#toc-1>
* <https://www.youtube.com/watch?v=NyOjKd5Qruk>
* Висновок:

У пам’яті комп’ютера існують дві основні області для розміщення даних:

* **Stack (стек):**
  + Використовується для зберігання локальних змінних і викликів функцій.
  + Пам'ять виділяється та звільняється автоматично.
  + Обмежена за розміром, але швидша у доступі.
* **Heap (купа):**
  + Використовується для виділення динамічної пам’яті.
  + Виділення та звільнення пам’яті контролюється програмістом (наприклад, через new/delete )
  + Гнучкий і підходить для динамічних структур даних , а також для роботи з складними структурами.

##### ****Особливості динамічного масиву:****

* Збільшення або зменшення розміру під час виконання програми.
* Ефективна робота зі змінними даними.
* Просте додавання та видалення елементів.
* ***2.Cтек:***
* Джерела:
* <https://dystosvita.org.ua/mod/page/view.php?id=888>
* <https://disted.edu.vn.ua/courses/learn/13472>
* Висновок:

**Стек** — це динамічна структура даних, яка працює за принципом **LIFO** (Last In, First Out): останній елемент, який додається до стеку, буде першим, що видаляється.

#### ****Операції зі стеком****

1. **Push:** Додає елемент у вершину стеку.
2. **Pop:** Видаляє елемент із вершини стеку.
3. **Top (або Peek):** Повертає елемент у вершині стеку без видалення.

**Приклади використання стеку:**

**Обернений польський запис**

***3. Черга:***

* Джерела:
* <https://dystosvita.org.ua/mod/page/view.php?id=889>
* Висновок:

**Черга (Queue)** — працює за принципом **FIFO (First In, First Out)**: елементи додаються в кінець черги, а видаляються з початку. Це означає, що перший елемент, який був доданий, буде першим видаленим.

-**enqueue**: додавання елемента в кінець черги.

-**dequeue**: видалення елемента з початку черги.

-**front**: перегляд елемента на початку черги.

***4. Зв'язні Списки:***

* Джерела:
* <https://prometheus.org.ua/cs50/sections/section6.html>
* Висновок:

**Зв'язний список** — це динамічна структура даних, яка складається з вузлів. Кожен вузол містить дані та вказівник на інший вузол.  
Види списків:

* **Однозв'язний список:**  
  Кожен вузол містить дані та вказівник на наступний вузол.
* **Двозв'язний список:**  
  Кожен вузол містить дані, вказівник на наступний вузол і вказівник на попередній вузол.

***5.*** ***Дерева:***

* Джерела:
* <https://purecodecpp.com/uk/archives/2483>
* <https://javarush.com/ua/groups/posts/uk.4165.chervono-chorne-derevo-vlastivost-principi-organzac-mekhanzmi-vstavki>

Висновок:

**Дерево** — це нелінійна структура даних, яка складається з вузлів, організованих у вигляді ієрархії.

* **Корінь дерева** (root): вузол, який є початком дерева.
* **Батьківські та дочірні вузли**: кожен вузол може мати дочірні вузли. Вузол, який має дочірній вузол, називається батьківським.
* **Листя** (leaf): вузли без дочірніх вузлів.

##### ****Типи дерев:****

1. **Бінарне дерево:** Кожен вузол має максимум два дочірніх вузли.
2. **Повне бінарне дерево:** Усі рівні, крім останнього, заповнені, а вузли останнього рівня розташовані зліва.
3. **Дерево пошуку (Binary Search Tree, BST):** Бінарне дерево, де для кожного вузла:
   * Значення всіх вузлів лівого піддерева менше, ніж значення вузла.
   * Значення всіх вузлів правого піддерева більше, ніж значення вузла.
4. **AVL-дерево:** Бінарне дерево пошуку, збалансоване за висотою.
5. **Червоно-чорне дерево:** Самобалансуюче бінарне дерево пошуку.

*Виконання роботи:*

Завдання №1

(VNS Lab10 Variant 6) -

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. Знищити з нього елемент із заданим номером, додати елемент у початок списку.

Завдання №2

(Algotester Lab5 V.2) -

В пустелі існує незвичайна печера, яка є двохвимірною. Її висота це NN, ширина - MM.

Всередині печери є пустота, пісок та каміння. Пустота позначається буквою ОО , пісок SS і каміння XX;

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.

# Input

У першому рядку 2 цілих числа NN та MM - висота та ширина печери

У NN наступних рядках стрічка rowirowi яка складається з NN цифер - i-й рядок матриці, яка відображає стан печери до землетрусу.

# Output

NN рядків, які складаються з стрічки розміром MM - стан печери після землетрусу.

# Constraints

1≤N,M≤10001≤N,M≤1000

|rowi|=M|rowi|=M

rowi∈{X,S,O}

Завдання №3

(Algotester Lab5 V.2) –

# Lab 78v1

*Limits: 2 sec., 256 MiB*

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

* **Вставка**:  
  Ідентифікатор - insertinsert  
  Ви отримуєте ціле число indexindex елемента, на місце якого робити вставку.  
  Після цього в наступному рядку рядку написане число NN - розмір списку, який треба вставити.  
  У третьому рядку NN цілих чисел - список, який треба вставити на позицію indexindex.
* **Видалення**:  
  Ідентифікатор - eraseerase  
  Ви отримуєте 2 цілих числа - indexindex, індекс елемента, з якого почати видалення та nn - кількість елементів, яку треба видалити.
* **Визначення розміру**:  
  Ідентифікатор - sizesize  
  Ви не отримуєте аргументів.  
  Ви виводите кількість елементів у списку.
* **Отримання значення** ii-го елементу  
  Ідентифікатор - getget  
  Ви отримуєте ціле число - indexindex, індекс елемента.  
  Ви виводите значення елемента за індексом.
* **Модифікація значення** ii-го елементу  
  Ідентифікатор - setset  
  Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення.
* **Вивід списку на екран**  
  Ідентифікатор - printprint  
  Ви не отримуєте аргументів.  
  Ви виводите усі елементи списку через пробіл.  
  Реалізувати використовуючи перегрузку оператора <<

# Notes

Гарантується, що усі дані коректні. Виходу за межі списку або розмір, більший ніж розмір списку недопустимі.  
Індекси починаються з нуля.  
  
**Для того щоб отримати**50%50%**балів за лабораторну достатньо написати свою структуру.  
Для отримання**100%100%**балів ця структура має бути написана як**[**шаблон класу**](https://cplusplus.com/doc/oldtutorial/templates/)**, у якості параметру використати**intint**.  
Використовувати STL заборонено.**

Завдання №4

(Class Practice Work)

## 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 №5

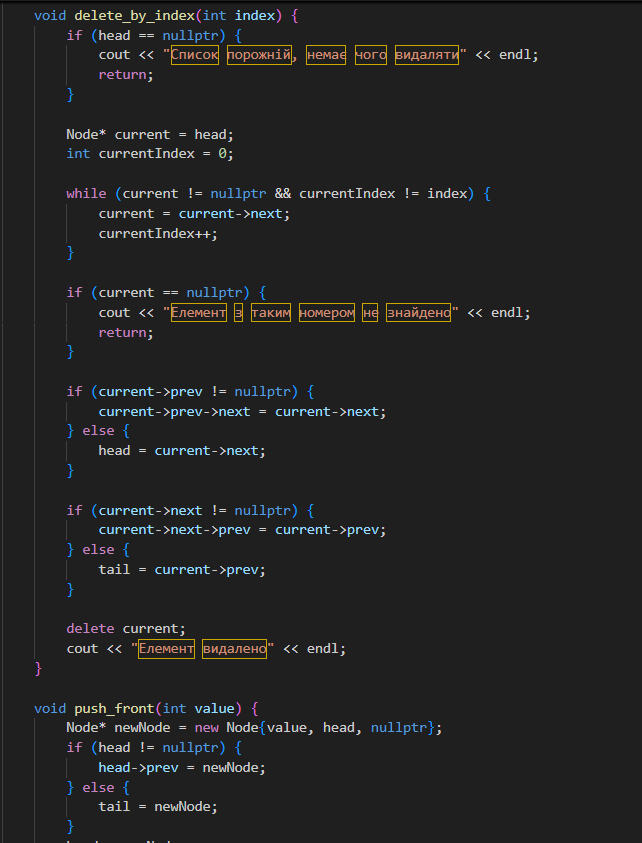
(Self Practice Work)

Реалізуй клас Library для управління динамічною колекцією книг із функціями додавання, видалення, пошуку, оновлення, виведення списку книг, зміни розміру масиву та підтримкою сортування і фільтрації за автором чи роком видання.

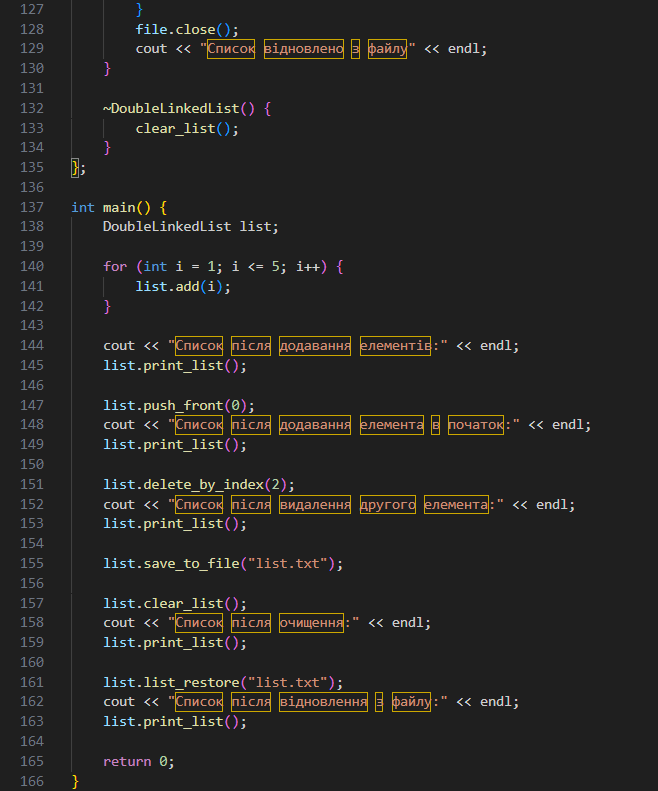
Завдання №1

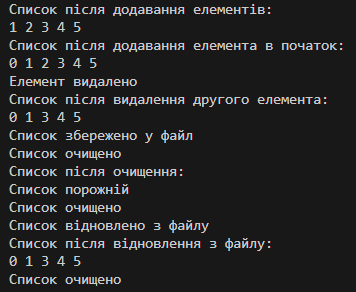
(VNS Lab10 Variant 6)





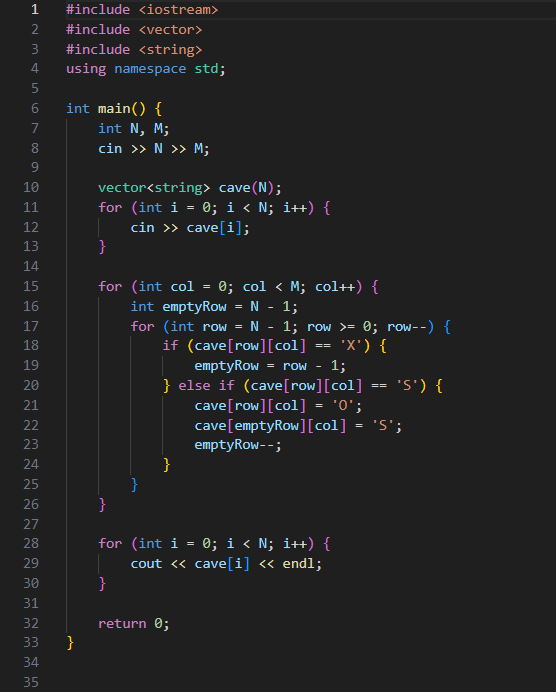


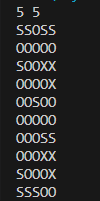


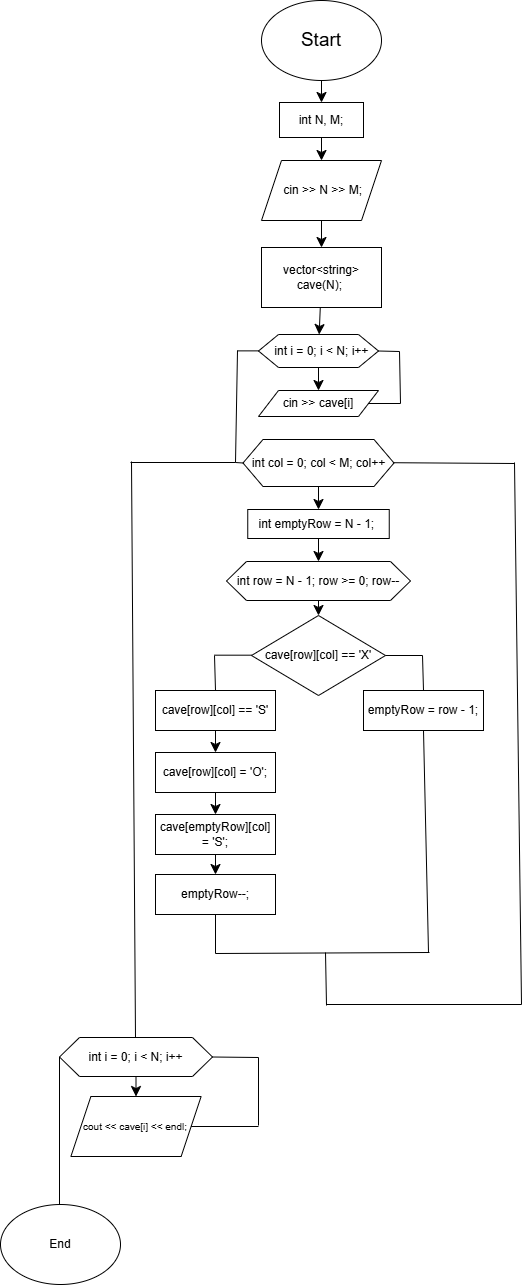


Завдання №2

(Algotester Lab5 V.2)



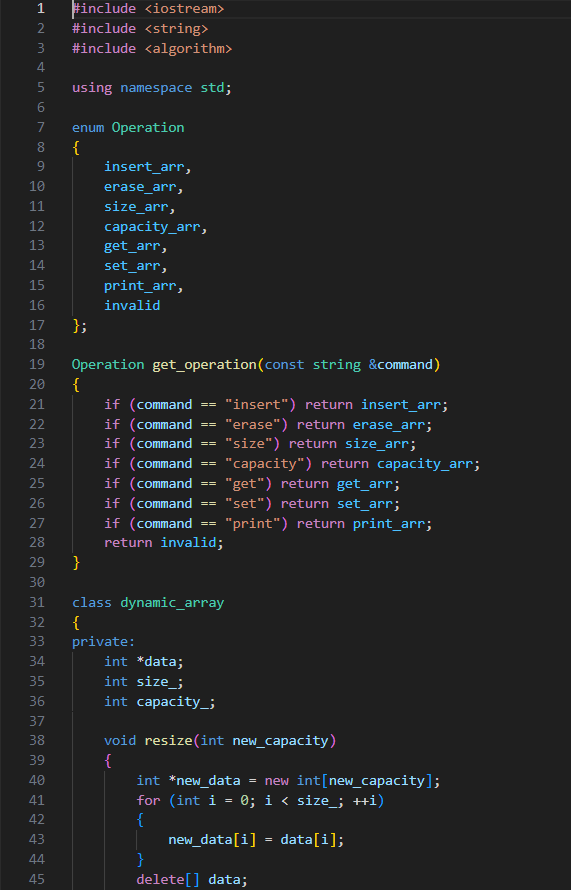


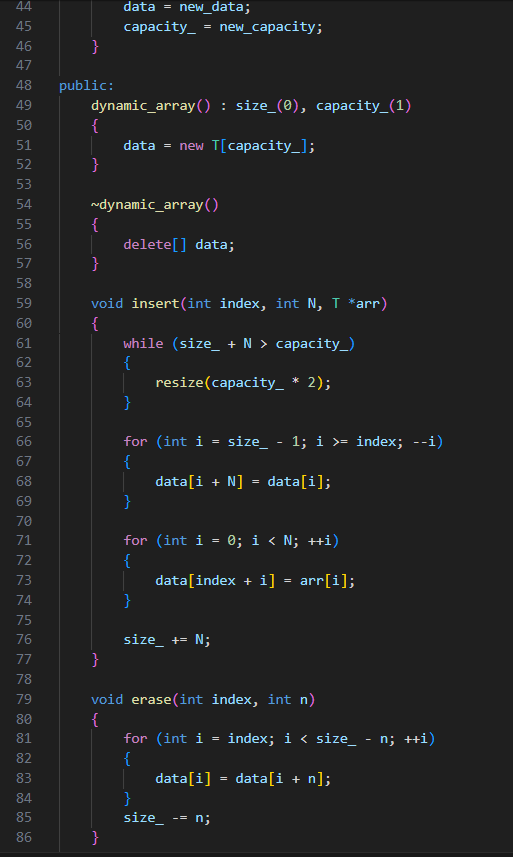


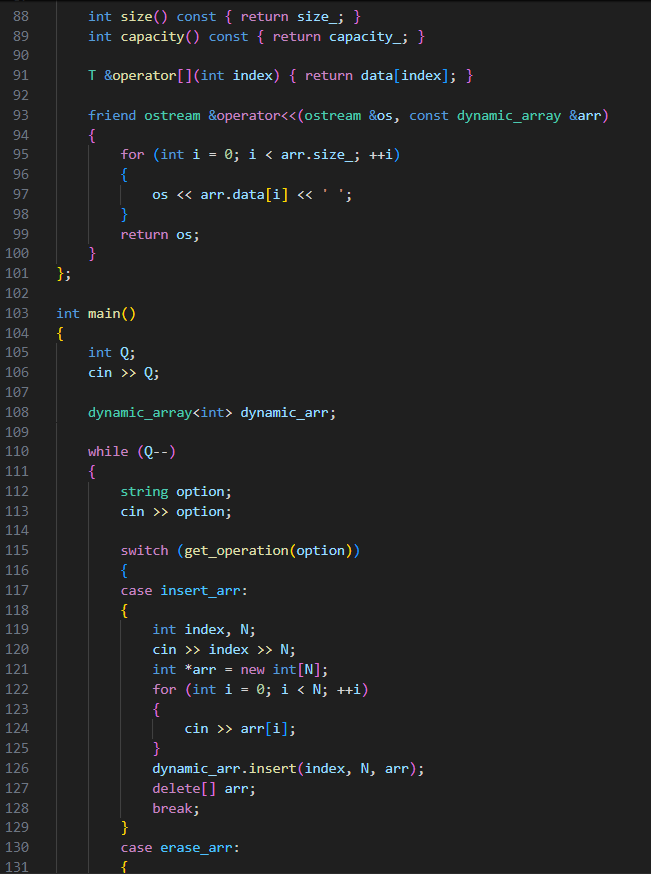
Завдання №3

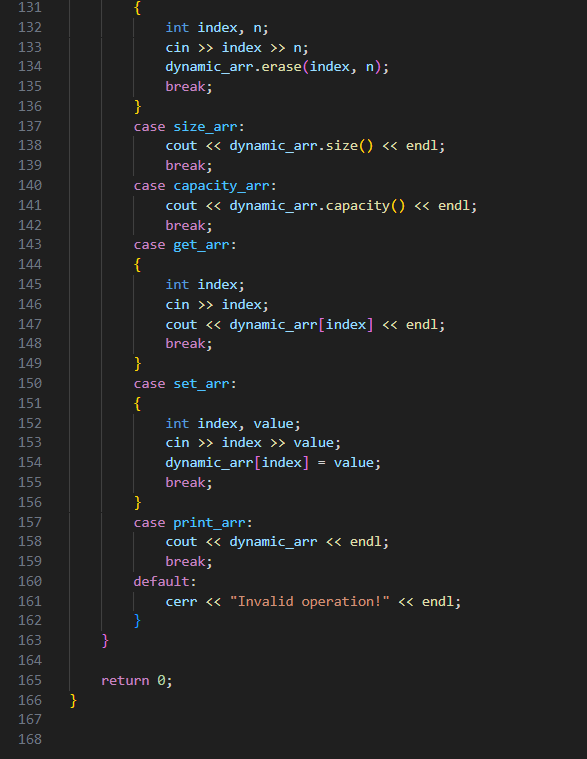
(VNS Lab7 Task1)

Варіант з написанням своєї структури:

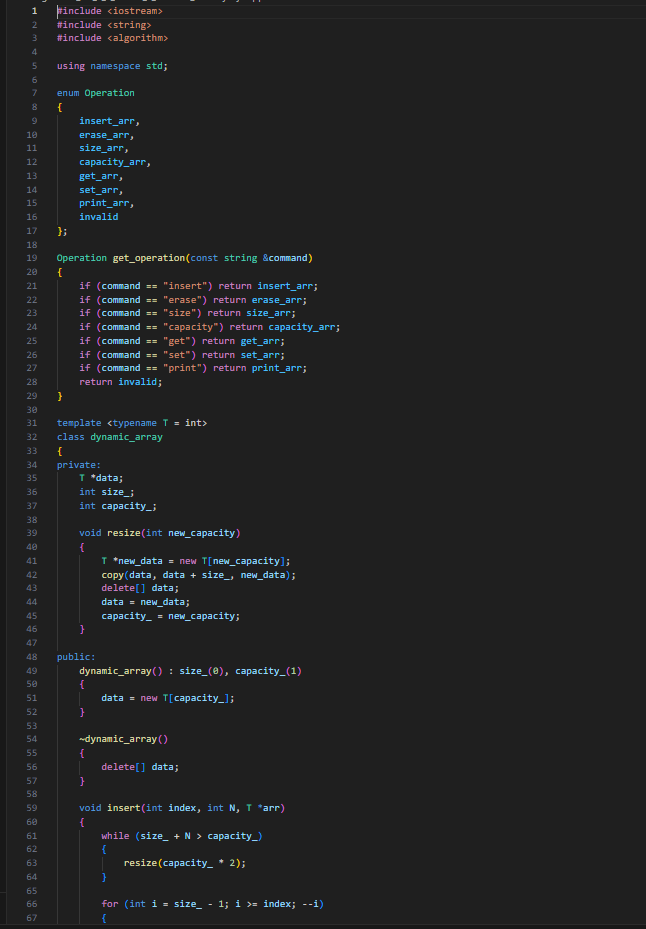


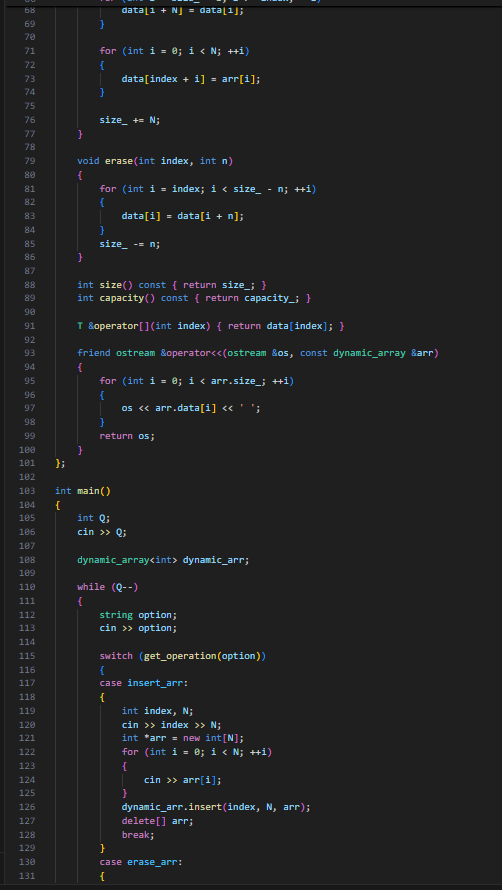


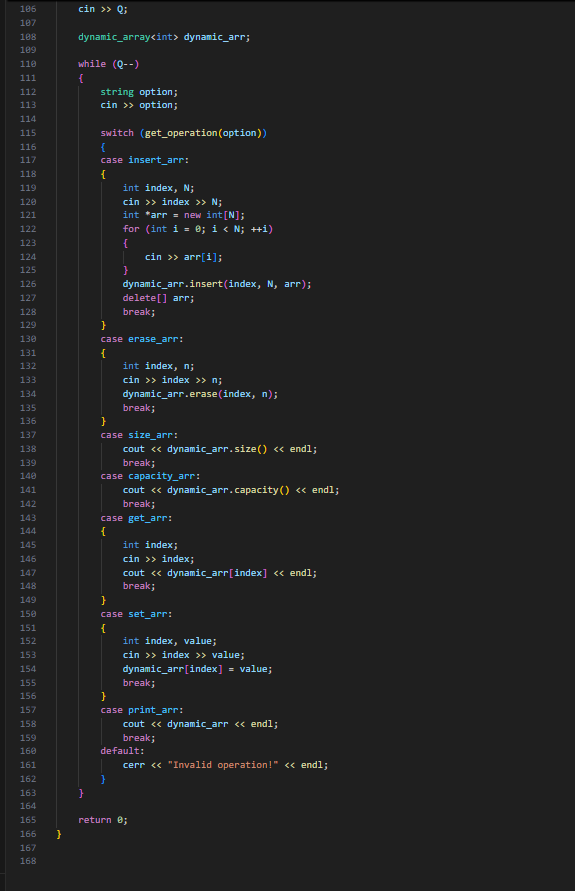


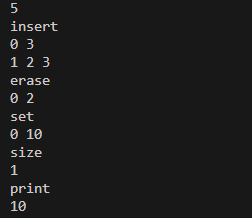


Структура написана як шаблон класу:





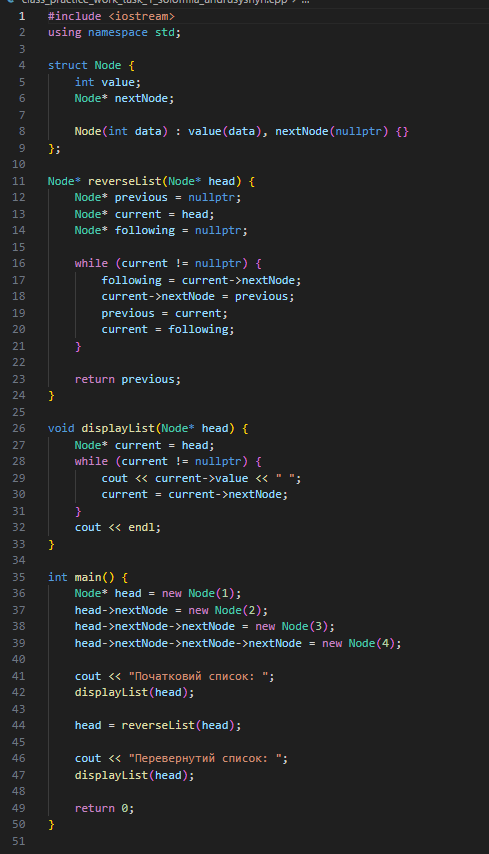




Завдання №4

(Class Practice Work)

1. **Task 1 - Реверс списку (Reverse list)**



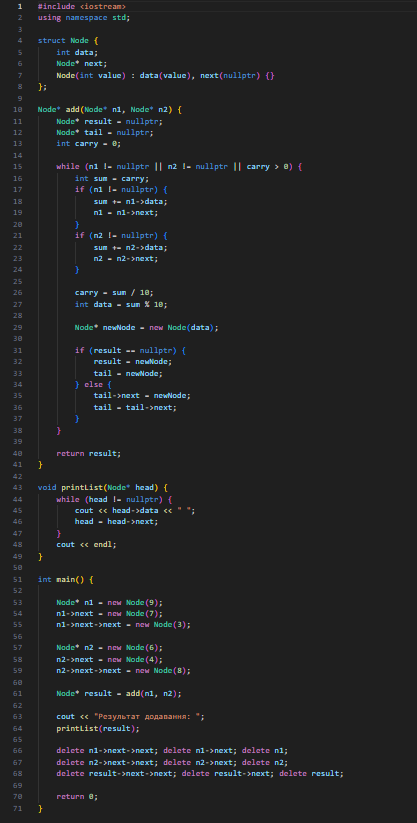


**Task 2 - Порівняння списків**



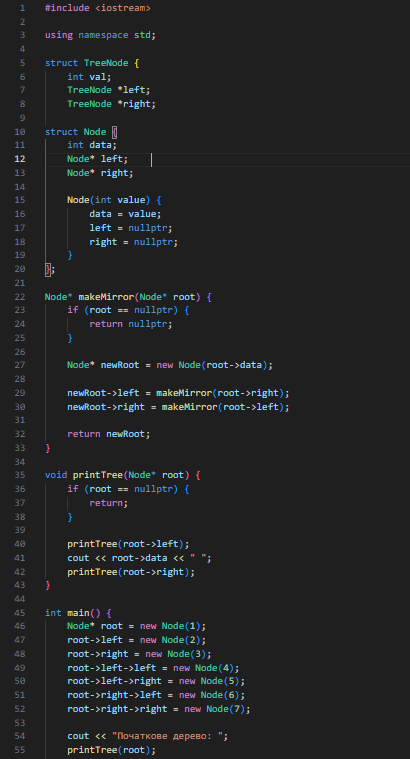


**Task 3 - Додавання великих чисел**





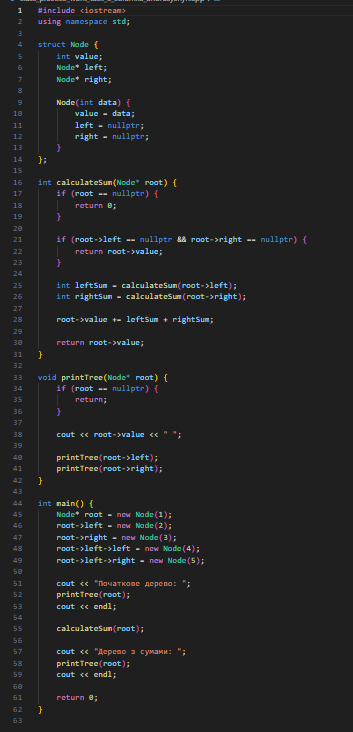
**Task 4 - Віддзеркалення дерева**







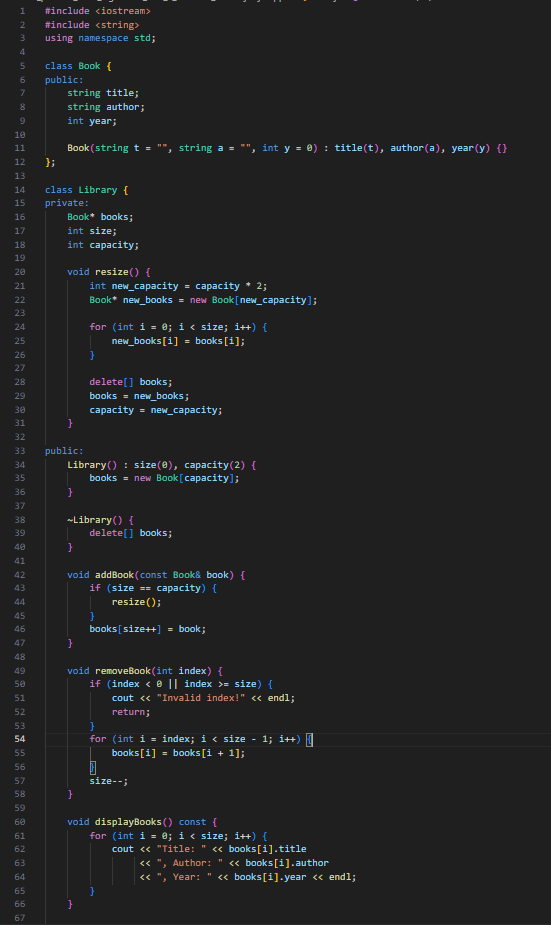
**Task 5 - Записати кожному батьківському вузлу суму підвузлів**

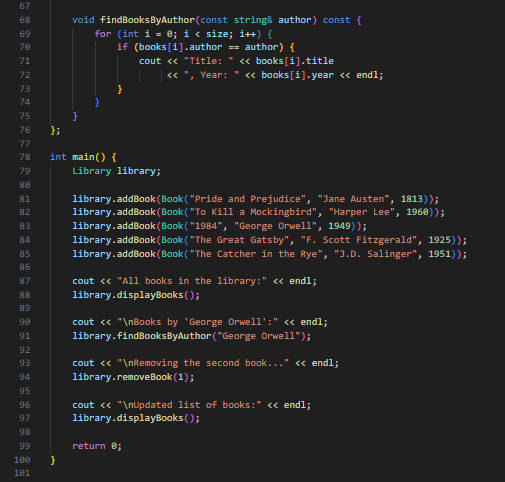


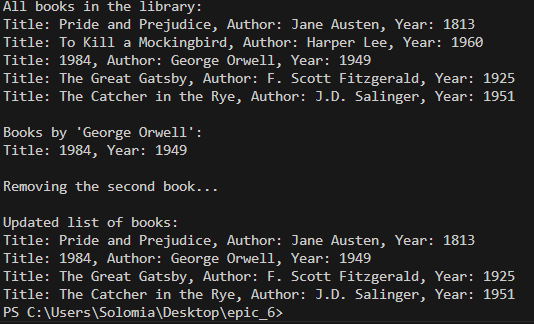


Завдання №5

(Self Practice Work)







*Висновок:* На лабораторній: роботі №6 (epic 6) , я ознайомилась з основними динамічними структурами даних у C++.Опанувала основні алгоритми пошуку, сортування, вставки та видалення елементів у цих структурах.