Лабораторна робота №1а
Моделювання з використанням UML

1)Код наведений у файлі: SplayTree.cpp

В наведеному коді реалізовано розширюване дерево, яке надає можливості розширення, пошуку, вставки, видалення елемента(наведені операції виконуються трьома методами: Zig, Zig-Zig та Zig-Zag). Також, за допомогою, даного коду можна переглядати вміст дерева та отримувати результат відносно ідентифікатора введених даних елемента в дереві.

Основні поняття та визначення:

Розширюване дерево (англ. *splay tree*) є двійковим деревом пошуку, у якому підтримується збалансованість. Це дерево належить до класу «саморегульованих дерев», які підтримують необхідний баланс галуження дерева, щоб забезпечити виконання операції пошуку, додавання і видалення за логарифмічний час від числа елементів, що зберігаються. Це реалізується без використання яких-небудь додаткових полів у вузлах дерева (як, наприклад, в Червоно-чорних деревах або АВЛ-деревах, де у вершинах зберігається, відповідно, колір вершини і глибина піддерева). Замість цього «розширювальні операції» (splay operation), частиною яких є повороти дерева, які виконуються при кожному зверненні до дерева.

Splay (розширення)
Основна операція дерева. Полягає в переміщенні вершини в корінь за допомогою послідовного виконання трьох, наведених нижче, операцій: Zig, Zig-Zig і Zig-Zag. Позначимо вершину, яку хочемо перемістити в корінь за x, її родича — p, а родича p (якщо існує) — g.

- **Zig:** виконується, коли p є коренем. Дерево повертається по ребру між x та p. Існує лише для розбору крайнього випадку і виконується лише один раз в кінці, коли початкова глибина x була непарна.

- **Zig-Zig:** виконується, коли x і p є лівими (або правими) синами. Дерево повертається по ребру між p та x.

- **Zig-Zag:** виконується, коли x є правим сином, а p — лівим (чи навпаки). Дерево повертається по ребру між p та x, а потім — по ребру між x та g.

Search (пошук елемента)
Пошук елемента в splay tree відбувається за допомогою операції splay, яка переміщує шуканий елемент на корінь дерева або на місце, де він повинен бути, якщо елемент не знайдено.

Insert (додавання елемента)

Додавання елемента в splay tree складається з кількох кроків, які включають пошук місця для нового вузла, вставку цього вузла та проведення операції splay для підтримання властивостей дерева.

Delete (видалення елемента)

Видалення елемента в дереві типу splay tree здійснюється в декілька етапів. Основна ідея полягає в тому, щоб видалити кореневий вузол, який був "розтягнутий" (splayed) до кореня.

Перегляд вмісту дерева:

Функція **printInOrder()** виконує перегляд вмісту дерева в порядку "впорядкованого" обходу (in-order traversal). Цей вид обходу дерева полягає в тому, що спочатку виводиться вміст лівого піддерева, потім виводиться поточний вузол, і, нарешті, виводиться вміст правого піддерева.Функція **inorder()** рекурсивно обходить дерево та виводить його вміст у правильному порядку. Починаючи з кореня, спочатку обробляється ліве піддерево (якщо воно існує), потім поточний вузол і, нарешті, праве піддерево (якщо воно існує).Коли ця функція викликається з кореневим вузлом, вона виводить вміст дерева в порядку, що відповідає правильному порядку обходу дерева. Таким чином, при виклику **printInOrder()** в консоль виводиться вміст дерева у відсортованому порядку, що є одним з основних властивостей splay tree.

Отримання ідентифікатора елемента:

Функція отримання ідентифікатора елемента реалізована у класі **SplayTree** за допомогою методу **search**. Отже, коли користувач вводить назву елемента для пошуку, програма використовує метод **search** для знаходження відповідного ідентифікатора, а потім повідомляє користувачеві про результат пошуку.

2) Unit tests, що описують функціональність:

Код наведений у файлі: UnitTest.mm

- ***Тести «testInsert» та «testSearch»:*** Перевіряємо правильність додавання елементів до дерева та їх подальшого пошуку. Спочатку додаємо кілька елементів до дерева. Перевіряємо, чи можна знайти елементи, які були додані. Та перевіряємо, чи неможливо знайти елемент, якого немає в дереві.

- ***Тест «testRemoveExistingElement»:*** Перевіряємо правильність видалення елементів з дерева. Для цього додаємо кілька елементів до дерева. Видаляємо один з цих елементів. Перевіряємо, чи був видалений елемент, тобто чи неможливо знайти його в дереві після видалення.

- ***Тест «testRemoveFromEmptyTree»:*** Перевіряємо, як працюють методи пошуку та видалення на порожньому дереві. Спочатку спробуємо знайти елемент та видалити його з порожнього дерева. Та перевіримо, що результати пошуку та видалення є відповідними порожнім деревам.

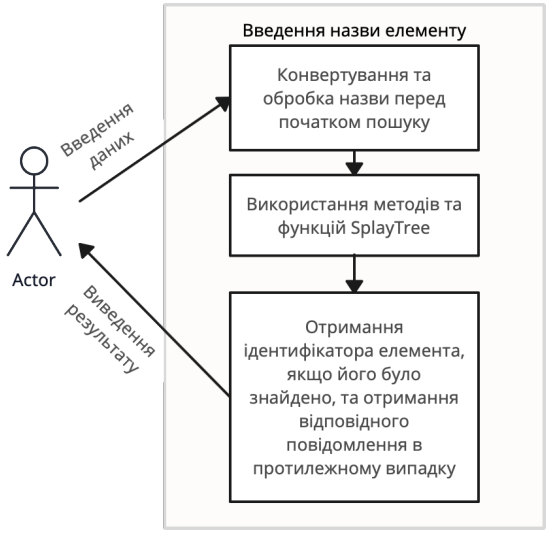
- ***Тест «testPrintTree»:*** Перевіряємо, чи правильно виводиться дерево в правильному порядку. Спочатку додаємо кілька елементів до дерева. Виводимо дерево за допомогою методу printInOrder. Перевіряємо, чи виведений результат співпадає з очікуваним порядком.

- ***Тест «testSearchByIdentifier»:*** Перевіряємо, чи правильно повертається ідентифікатор елемента за його назвою. Спочатку додаємо кілька елементів до дерева. Здійснюємо пошук ідентифікатора за назвою певного елемента. Перевіряємо, чи повертається правильний ідентифікатор для знайденого елемента.

Після виконання вищенаведених тестів, реалізація всіх методів та функцій SplayTree працює коректно.

3) UML діаграми, що описують SplayTree:

Use Case: програма виконує пошук за введеними користувачем даними та виводить отриманий результат.



Структура коду (Class, Component, Object, Composite Structure, Deployment, Package):

Class: SplayTree, Node

Component: main, root, insert, remove, search, rotateLeft, rotateRight;

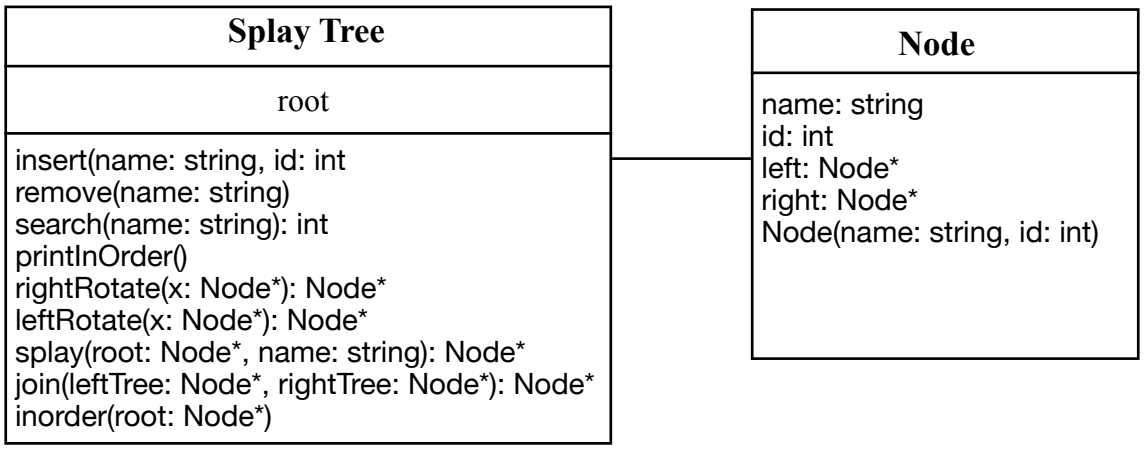
Object: SplayTree

Composite Structure: Розширюване дерево

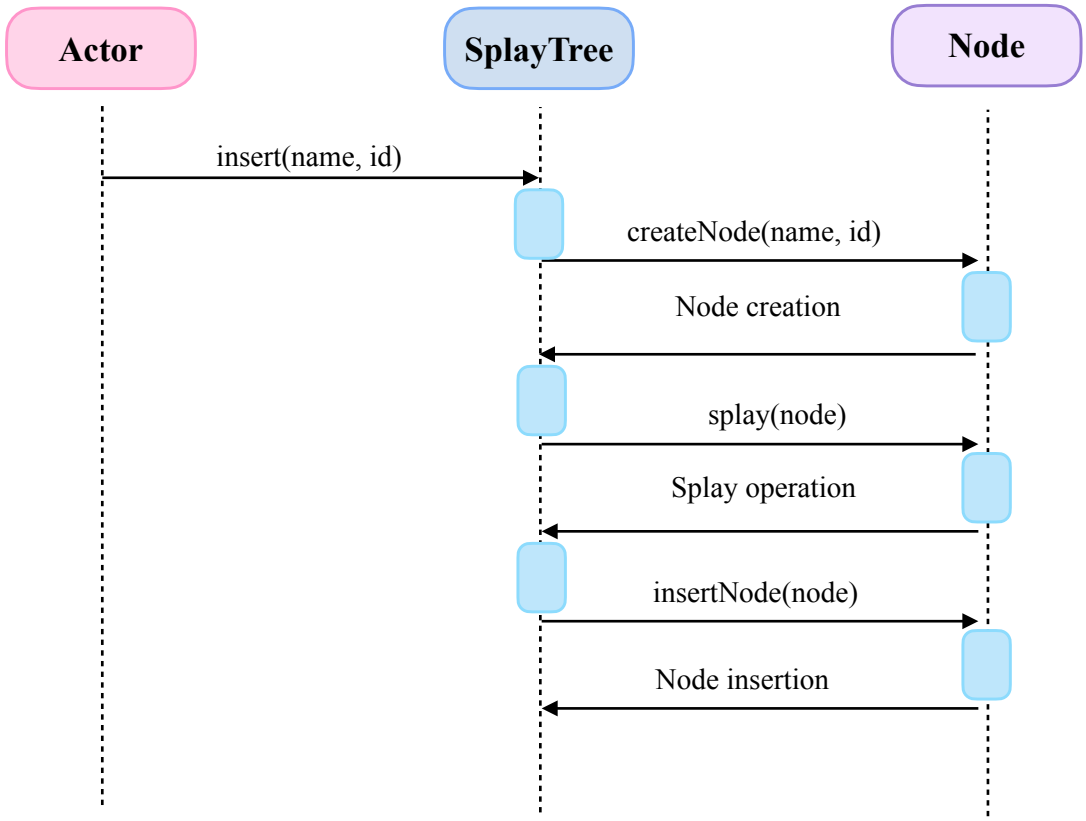
Deployment: виклик методів та функцій

Package: відсутні

Class Diagram



Логіка та поведінка програми (Sequence, Communication, Timing, Activity, Interaction Overview, State)



Sequence Diagram для операції вставки елемента

4)Запропонувати зміни в структурі/інтерфейсі/реалізації програми:

Перехід до ООП:

- Розділення класів: Клас SplayTree може бути розділений на кілька менших класів для покращення читабельності та підтримки. Наприклад, можна створити окремий клас для вузлів (Node) і для операцій над деревом (SplayTree).

- Інкапсуляція: Можливо зробити усі члени класу приватними, надаючи доступ до них через методи. Це покращить інкапсуляцію даних.

Поділ на компоненти:

- Інтерфейс користувача: Логіка роботи з користувачем має бути винесена у окремий клас або функції. Це дозволить легше змінювати інтерфейс без впливу на основну логіку дерева.

Використання патернів проєктування:

- Factory Method: Патерн Factory Method може бути корисним для створення вузлів дерева. Це дозволить легше модифікувати процес створення вузлів у майбутньому.

- Патерн Singleton: Якщо потрібно мати лише один екземпляр дерева в програмі, можна застосувати патерн Singleton.

Інтерфейс:

Інтерфейс для дерева: Може бути створений інтерфейс Tree з основними методами (insert, remove, search, printInOrder). Це дозволить легко замінювати реалізацію дерева на іншу (наприклад, AVL дерево) без зміни основного коду.

Також, корисним для використання може бути додання методів для зберігання даних в файл та їх завантаження з файлу.

5)Реалізувати запропоновані зміни:

Код наведений у файлі: Modify_SplayTree.cpp

6)Додані зміни в методи для збереження та завантаження даних не змінюють логіку існуючих методів роботи з деревом. Всі основні алгоритми для основних можливих операцій Splay Tree : вставки, видалення, пошуку та обходу дерева залишилися без змін та додаткові методи для роботи з файлами працюють незалежно від основних операцій з деревом, що дозволяє не впливати на них. Отже, оновлена версія коду не вносить змін в логіку чи алгоритми роботи з розширюваним деревом.

7) Порівняти попередню та оновлену версії програми за часом виконання окремих алгоритмів/функцій, обсягом коду і т.д.

Час виконання: Оскільки основні алгоритми для роботи з деревом (вставка, видалення, пошук, обхід) не були змінені, отже й час виконання цих функцій залишився таким самим, як і в початковій версії.

Обсяг коду: До оновленої програми додано нові методи для зберігання та завантаження дерева з файлу, що призвело до збільшення загального обсягу коду. Але основну логіку роботи з деревом це не змінило.

Функціональність: До оновленої програми було додано нову функціональність для роботи з файлами, що дозволяє зберігати та відновлювати стан дерева між сесіями.

Отже, оновлена версія програми зберігає логіку та алгоритми основних операцій, але надає додаткову функціональність без значного впливу на ефективність.

8) Проаналізувати код (попередній, змінений, самі зміни):

Принципи архітектури та проєктування	Початковий код	Оновлений код
Інкапсуляція	Члени класу закриті, але логіка дерева не розділена	Поля і методи розташовані відповідно до принципу Інкапсуляції, додано новий клас **Tree**
Успадкування	Відсутнє	Введено успадкування, **SplayTree** успадковує **Tree**

Поліморфізм	Відсутній	Застосовано віртуальні методи у базовому класі **Tree**
KISS	Проста структура, але може бути складною для розширення	Додано новий клас та методи, але збережено просту структуру
Можливість розвитку	Деякою мірою складний для розширення	Зроблено більш гнучким і легко розширюваним за рахунок введення класу **Tree**
YAGNI	Код може містити зайвий функціонал	Додано новий функціонал, який може знадобитися в майбутньому
SOLID	Відсутній	Застосовано частково, зокрема принципи: принцип однієї відповідальності та принцип відкритості/закритості
Coupling/Cohesion	Не враховано	Введення нового класу може покращити зв'язність
Law of Demeter	Можливе порушення	Дотримано завдяки введенню базового класу та збереженню керованої зв'язності
Патерни	Не використовуються	Factory Method та Singleton