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:

ВНС Лабораторної Роботи № 10

Алготестер Лабораторної Роботи № 5

Алготестер Лабораторної Роботи № 7-8

Практичних Робіт до блоку № 6

Виконала:

Студентка групи ШІ-11

Гуменюк Анастасія Олександрівна

Львів 2024

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:

Теоретичні відомості з переліком важливих тем:

- Тема №1: Основи Динамічних Структур Даних.
- Тема №2: Стек.
- Тема №3: Черга.
- Тема №4: Зв'язні Списки.
- Тема №5: Дерева.

Індивідуальний план опрацювання теорії:

Тема №1: Основи Динамічних Структур Даних.

- Джерела:
  https://acode.com.ua/urok-111-stek-i-kupa/#toc-1
  https://www.youtube.com/watch?v=NyOjKd5Qruk
- Що опрацьовано:
  - Вступ до динамічних структур даних: визначення та важливість
  - Виділення пам'яті для структур даних (stack і heap)
  - Приклади простих динамічних структур: динамічний масив
- Статус: Ознайомлена
- Початок опрацювання теми: 25.11.2024.
- Звершення опрацювання теми: 25.11.2024 (20 хв.).

Тема №2: Стек.

- Джерела:
  https://dystosvita.org.ua/mod/page/view.php?id=888

https://disted.edu.vn.ua/courses/learn/13472
https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D1%81%D1%8C%D0%BA%D0%B8%D0%B9_%D1%96%D0%BD%D0%B2%D0%B5%D1%80%D1%81%D0%BD%D0%B8%D0%B9_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81

- Що опрацьовано:
    - Визначення та властивості стеку
    - Операції push, pop, top: реалізація та використання
    - Приклади використання стеку: обернений польський запис
    - Переповнення стеку

- Статус: Ознайомлена
- Початок опрацювання теми: 26.11.2024.
- Звершення опрацювання теми: 26.11.2024 (30 хв.).

Тема №3: Черга.

- Джерела:
  https://dystosvita.org.ua/mod/page/view.php?id=889
  https://www.kostrub.online/2020/07/struktura-danyx-cherha-Queue.html
- Що опрацьовано:
    - Визначення та властивості черги
    - Операції enqueue, dequeue, front: реалізація та застосування
    - Приклади використання черги: обробка подій, алгоритми планування
    - Розширення функціоналу черги: пріоритетні черги
    - Статус: Ознайомлена
- Початок опрацювання теми: 26.11.2024.
- Звершення опрацювання теми: 26.11.2024 (25 хв.).

Тема №4: Зв'язні Списки.

- Джерела:
  https://prometheus.org.ua/cs50/sections/section6.html
- Що опрацьовано:
    - 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

- Статус: Ознайомлена
- Початок опрацювання теми: 27.11.2024.
- Звершення опрацювання теми: 27.11.2024 (1 год 23 хв.).

Тема №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
- Що опрацьовано:
  - 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
- Статус: Ознайомлена
- Початок опрацювання теми: 27.11.2024.
- Звершення опрацювання теми: 27.11.2024 (1 год 23 хв.).

Виконання роботи:

1) Опрацювання завдання та вимог до програми та середовища

Завдання №1 - VNS Lab 10 - Task 1-10

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

Обмеження: 2 сек., 256 МіБ

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 і-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 N 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 N - кількість заклинань.

У наступних N рядках по 2 числа Ri, Ci - рядок і стовпець кулі над якою виконується заклинання.

Вихідні дані

Одне число b - стан дошки після виконання N заклинань.

Обмеження

$0 \le N \le 10^3$

$1 \le Ri, Ci \le 8$

$0 \le a, b < 2^{64}$

Завдання №3 - Algotester Lab 7-8

Обмеження: 1 сек., 256 МіБ

Ваше завдання - власноруч реалізувати структуру даних "Двійкове дерево пошуку".
Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його параметри.

Вам будуть поступати запити такого типу:

- **Вставка**:
  Ідентифікатор - insert
  Ви отримуєте ціле число value - число, яке треба вставити в дерево.
- **Пошук**:
  Ідентифікатор - contains
  Ви отримуєте ціле число valuevalue - число, наявність якого у дереві необхідно перевірити.
  Якщо value наявне в дереві - ви виводите Yes, у іншому випадку No.
- **Визначення розміру**:
  Ідентифікатор - size

Ви не отримуєте аргументів.
Ви виводите кількість елементів у дереві.

- **Вивід дерева на екран**
  Ідентифікатор - print
  Ви не отримуєте аргументів.
  Ви виводите усі елементи дерева 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.

Обмеження

$0 \le Q \le 10^3$
$0 \le ti \le 10^3$

Завдання №4 - Class Practice Work - Task 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.

Завдання №5 - Class Practice Work - Task 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.

Завдання №6 - Class Practice Work - Task 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.

Завдання №7 - Class Practice Work - Task 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.

Завдання №8 - Class Practice Work - Task 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 №9 - Self Practice Work – Черга

Умова задачі:

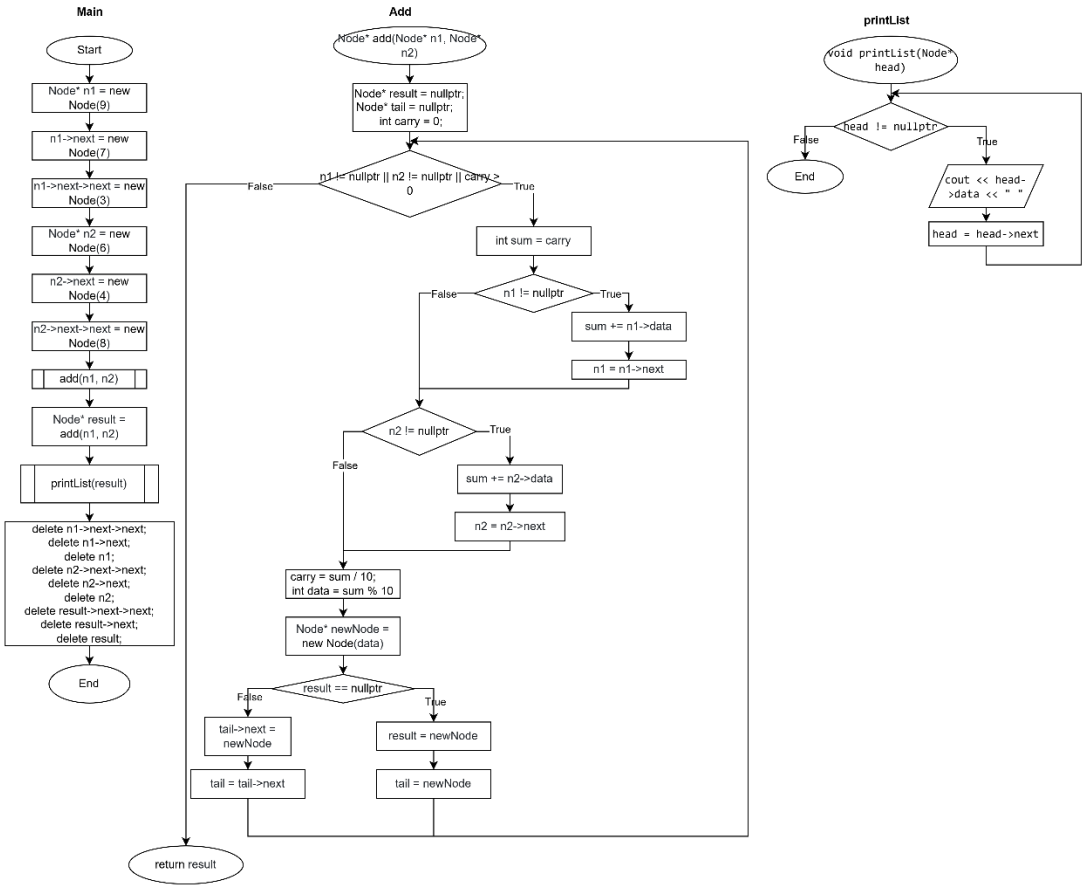
У вас є черга, яка реалізована за допомогою зв'язного списку. Реалізуйте клас Queue, що підтримує стандартні операції черги: додавання елементів (enqueue), видалення елементів (dequeue), перевірку розміру черги та виведення елементів черги.

Опис класу Queue:

1. Клас має приватну структуру Node, яка представляє окремий елемент черги, що містить ціле значення data та вказівник на наступний елемент next.
2. Клас має два вказівники: frontNode (вказівник на перший елемент черги) та rearNode (вказівник на останній елемент черги).
3. Операції:
    - enqueue(int element) — додає новий елемент у кінець черги.
    - dequeue() — видаляє елемент з початку черги, якщо вона не порожня.
    - getSize() — повертає кількість елементів у черзі.
    - printQueue() — виводить всі елементи черги.
4. Якщо черга порожня і виконується операція dequeue(), вивести повідомлення: "Черга порожня. Неможливо видалити елемент."
5. Реалізувати деструктор, який очищує всю пам'ять, зайняту чергою.

2) Дизайн виконання завдань:

Завдання №6 - Class Practice Work - Task 3 - Додавання великих чисел



3) Код програм з посиланням на зовнішні ресурси та фактично затрачений час:

Завдання №1 - VNS Lab 10 - Task 1-10

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

struct Node {
    int data;
    Node* prev;
    Node* next;

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

void pushToEnd(Node*& head, Node*& tail, int value) {
    Node* newNode = new Node(value);

    if (head == nullptr) {
        head = tail = newNode;
    } else {
```

```
        tail->next = newNode;
        newNode->prev = tail;
        tail = newNode;
    }
}

void insert(Node*& head, Node*& tail, int pos, int value) {
    Node* newNode = new Node(value);

    if (pos == 0 || head == nullptr) {
        newNode->next = head;
        if (head)
            head->prev = newNode;
        head = newNode;
        if (tail == nullptr)
            tail = head;
        return;
    }

    Node* current = head;
    for (int i = 0; current && i < pos - 1; i++) {
        current = current->next;
    }

    if (current == nullptr) {
        pushToEnd(head, tail, value);
        return;
    }

    newNode->next = current->next;
    newNode->prev = current;

    if (current->next)
        current->next->prev = newNode;
    current->next = newNode;

    if (!newNode->next)
        tail = newNode;
}

void deleteFromEnd(Node*& head, Node*& tail, int k) {
    while (k > 0 && tail) {
        Node* temp = tail;
        tail = tail->prev;

        if (tail) {
            tail->next = nullptr;
        } else {
            head = nullptr;
        }
        delete temp;
```

```
        k--;
    }
}

void printList(Node* head) {
    if (!head) {
        cout << "List is empty" << endl;
        return;
    }
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

void writeListToFile(Node* head, const string& filename) {
    ofstream file(filename, ios::out);
    if (!file) {
        cout << "Can not open the file!" << endl;
        return;
    }

    while (head) {
        file << head->data << " ";
        head = head->next;
    }

    file.close();
    cout << "List was written." << endl;
}

void deleteList(Node*& head) {
    while (head) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
}

void restore(Node*& head, Node*& tail, const string& filename) {
    ifstream file(filename, ios::in);
    if (!file) {
        cout << "Can not open the file!" << endl;
        return;
    }

    int value;
    while (file >> value) {
        pushToEnd(head, tail, value);
    }
```

```
    file.close();
    cout << "List was restored" << endl;
}

int main() {
    Node* head = nullptr;
    Node* tail = nullptr;

    pushToEnd(head, tail, 1);
    pushToEnd(head, tail, 2);
    pushToEnd(head, tail, 3);
    pushToEnd(head, tail, 4);

    cout << "List: ";
    printList(head);

    insert(head, tail, 2, 99);
    cout << "List after insertion 99 at second placr: ";
    printList(head);

    deleteFromEnd(head, tail, 2);
    cout << "List after removal 2 elements: ";
    printList(head);

    writeListToFile(head, "list.txt");

    deleteList(head);
    tail = nullptr;

    cout << "List after free: ";
    printList(head);

    restore(head, tail, "list.txt");
    cout << "List after restore: ";
    printList(head);

    deleteList(head);
    tail = nullptr;

    return 0;
}
```

Планований час: 1 год 30 хв. Фактичний: 2 год.

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/479/files#diff-072998404d072a8a37c2d9eb4484cc3531667717ce4517f3e626bad306653a70

Завдання №2 - Algotester Lab 5

```cpp
#include <iostream>

using namespace std;

int main() {
    unsigned long long a;
    int N;
    cin >> a >> N;

    for (int i = 0; i < N; i++) {
        int R, C;
        cin >> R >> C;

        R -= 1;
        C -= 1;

        for (int col = 0; col < 8; col++) {
            a ^= (1ULL << (R * 8 + col));
        }

        for (int row = 0; row < 8; row++) {
            a ^= (1ULL << (row * 8 + C));
        }

        a ^= (1ULL << (R * 8 + C));
    }

    cout << a << endl;

    return 0;
}
```

Планований час: 30 хв. Фактичний: 1 год.

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/479/files#diff-c4ae3ea3e272d13a06cfa6b6bbaedd1c3bacd9e6e6cf91e74682994779e1a4b1

Завдання №3 - Algotester Lab 7-8

```cpp
#include <iostream>

using namespace std;

template<typename T>
struct TreeNode {
    T data;
    TreeNode* left;
    TreeNode* right;
```

```
    TreeNode(T value) : data(value), left(nullptr), right(nullptr) { }
};

template<typename T>
class BinarySearchTree {
public:
    // Constructor
    BinarySearchTree() : root(nullptr), treeSize(0) {}

    void insert(T value) {
        root = insert(root, value);
    }

    bool contains(T value) {
        return contains(root, value);
    }

    int size() {
        return treeSize;
    }

    void print() {
        if (root == nullptr) {
            cout << endl;  // Якщо дерево порожнє
        } else {
            inorderTraverse(root);
            cout << endl;
        }
    }

private:
    TreeNode<T>* root;
    int treeSize;

    TreeNode<T>* insert(TreeNode<T>* node, T value) {
        if (node == nullptr) {
            treeSize++;
            return new TreeNode<T>(value);
        }

        if (value < node->data) {
            node->left = insert(node->left, value);
        } else if (value > node->data) {
            node->right = insert(node->right, value);
        }

        return node;
    }

    bool contains(TreeNode<T>* node, T value) {
        if (node == nullptr) {
```

```
            return false;
        }

        if (value == node->data) {
            return true;
        } else if (value < node->data) {
            return contains(node->left, value);
        } else {
            return contains(node->right, value);
        }
    }

    void inorderTraverse(TreeNode<T>* node) {
        if (node != nullptr) {
            inorderTraverse(node->left);
            cout << node->data << " ";
            inorderTraverse(node->right);
        }
    }
};

int main() {
    int Q;
    cin >> Q;

    BinarySearchTree<int> tree;
    string command;
    int value;

    for (int i = 0; i < Q; i++) {
        cin >> command;

        if (command == "insert") {
            cin >> value;
            tree.insert(value);
        } else if (command == "contains") {
            cin >> value;
            if (tree.contains(value)) {
                cout << "Yes" << endl;
            } else {
                cout << "No" << endl;
            }
        } else if (command == "size") {
            cout << tree.size() << endl;
        } else if (command == "print") {
            tree.print();
        }
    }

    return 0;
}
```

Планований час: 1 год 30 хв. Фактичний: 1 год 30 хв.

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/479/files#diff-02ae60b06dfde1bffa6094446fa5d638d7e42794564959aa1e2ab5bd7befa8bf

Завдання №4 - Class Practice Work - Task 1 - Реверс списку (Reverse list)

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

struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

Node* reverse(Node* head) {
    Node* prev = nullptr;
    Node* curr = head;
    Node* next = nullptr;

    while (curr != nullptr) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }

    return prev;
}

void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {

    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head->next->next->next = new Node(4);
```

```
    //Node* head = new Node(1, new Node(2, new Node(3, new Node(4,nullptr))));
    cout << "Вихідний список: ";
    printList(head);

    head = reverse(head);

    cout << "Обернений список: ";
    printList(head);

    return 0;
}
```

Планований час: 40 хв. Фактичний: 40 хв.

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/479/files#diff-ec18442adba0014e41a68e04914082075f510f9efa1670b821a1dc5977a01e9b

Завдання №5 - Class Practice Work - Task 2 - Порівняння списків

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

struct Node {
    int data;
    Node* next;

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

bool compare(Node* h1, Node* h2) {

    while (h1 != nullptr && h2 != nullptr) {
        if (h1->data != h2->data) {
            return false;
        }
        h1 = h1->next;
        h2 = h2->next;
    }

    return h1 == nullptr && h2 == nullptr;
}

void push(Node*& head, int value) {
    Node* newNode = new Node(value);
    newNode->next = head;
    head = newNode;
}

void pushToEnd(Node*& head, int value) {
    Node* newNode = new Node(value);
```

```
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

int main() {
    Node* head1 = nullptr;
    pushToEnd(head1, 1);
    pushToEnd(head1, 2);
    pushToEnd(head1, 3);

    Node* head2 = nullptr;
    push(head2, 3);
    push(head2, 2);
    push(head2, 1);

    if (compare(head1, head2)) {
        cout << "Списки однакові" << endl;
    } else {
        cout << "Списки різні" << endl;
    }

    delete head1;
    delete head2;

    return 0;
}
```

Планований час: 1 год. Фактичний: 1 год.

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/479/files#diff-68dbe4414126c2245ce5956f19e578ecf1ce3d03c1a660f69aecf25e36ddc627

Завдання №6 - Class Practice Work - Task 3 - Додавання великих чисел

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

struct Node {
    int data;
    Node* next;
```

```
    Node(int value) : data(value), next(nullptr) {}
};

Node* add(Node* n1, Node* n2) {
    Node* result = nullptr;
    Node* tail = nullptr;
    int carry = 0; // перенос розряду

    while (n1 != nullptr || n2 != nullptr || carry > 0) {
        int sum = carry;
        if (n1 != nullptr) {
            sum += n1->data;
            n1 = n1->next;
        }
        if (n2 != nullptr) {
            sum += n2->data;
            n2 = n2->next;
        }

        carry = sum / 10;
        int data = sum % 10;

        Node* newNode = new Node(data);

        if (result == nullptr) {
            result = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            tail = tail->next;
        }
    }

    return result;
}

void printList(Node* head) {
    while (head != nullptr) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

int main() {

    Node* n1 = new Node(9);
    n1->next = new Node(7);
    n1->next->next = new Node(3);

    Node* n2 = new Node(6);
```

```
    n2->next = new Node(4);
    n2->next->next = new Node(8);

    Node* result = add(n1, n2);

    cout << "Результат додавання: ";
    printList(result);

    delete n1->next->next; delete n1->next; delete n1;
    delete n2->next->next; delete n2->next; delete n2;
    delete result->next->next; delete result->next; delete result;

    return 0;
}
```

Планований час: 1 год 30 хв. Фактичний: 1 год 30 хв.

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/479/files#diff-24f809eb484afc2a0dd97efd16690292d0ce0bd921808acce275f230f1d46d6b

Завдання №7 - Class Practice Work - Task 4 - Віддзеркалення дерева

```
#include <iostream>

using namespace std;

struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

TreeNode* create_mirror(TreeNode* root) {
    if (root == nullptr) {
        return nullptr;
    }

    TreeNode* newNode = new TreeNode(root->val);

    newNode->left = create_mirror(root->right);
    newNode->right = create_mirror(root->left);

    return newNode;
}

void print(TreeNode* root) {
    if (root == nullptr) return;
```

```
    print(root->left);
    cout << root->val << " ";
    print(root->right);
}

int main() {

    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(7);

    cout << "Original tree: ";
    print(root);
    cout << endl;

    TreeNode* mirroredRoot = create_mirror(root);

    cout << "Mirrored tree: ";
    print(mirroredRoot);
    cout << endl;

    return 0;
}
```

Планований час: 40 хв. Фактичний: 45 хв.

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/479/files#diff-d15fc6d8da71839cb0a3b29f0623f128fd86fcb009183d065110fa49aae6dabd

Завдання №8 - Class Practice Work - Task 5 - Записати кожному батьківському вузлу суму підвузлів

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

struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int x) : data(x), left(nullptr), right(nullptr) {}
};

int calculate_sum(TreeNode* root) {
```

```
    if (!root)
        return 0;

    if (!root->left && !root->right) //вузол - листок
        return root->data;

    int left_sum = calculate_sum(root->left);
    int right_sum = calculate_sum(root->right);

    root->data += left_sum + right_sum;

    return root->data;
}

void print_tree(TreeNode* root) {
    if (!root) return;
    cout << root->data << " ";
    print_tree(root->left);
    print_tree(root->right);
}

int main() {

    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);

    cout << "Original tree: ";
    print_tree(root);
    cout << endl;

    calculate_sum(root);

    cout << "Sum tree: ";
    print_tree(root);
    cout << endl;

    return 0;
}
```

Планований час: 40 хв. Фактичний: 40 хв.

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/479/files#diff-1a064004ed0502c8078a4244dc52e88af0c900354873ad28fc45b767075ba228

Завдання №9 - Self Practice Work – Черга

```
#include <iostream>
```

```cpp
using namespace std;

class Queue {
private:
    struct Node {
        int data;
        Node* next;
        Node(int val) : data(val), next(nullptr) {}
    };

    Node* frontNode;
    Node* rearNode;

    bool isEmpty() const {
        return frontNode == nullptr;
    }

    int size() const {
        int count = 0;
        Node* current = frontNode;
        while (current != nullptr) {
            count++;
            current = current->next;
        }
        return count;
    }

public:
    Queue() : frontNode(nullptr), rearNode(nullptr) {}

    void enqueue(int element) {
        Node* newNode = new Node(element);
        if (isEmpty()) {
            frontNode = rearNode = newNode;
        } else {
            rearNode->next = newNode;
            rearNode = newNode;
        }
    }

    void dequeue() {
        if (!isEmpty()) {
            Node* temp = frontNode;
            frontNode = frontNode->next;
            delete temp;

            if (frontNode == nullptr) {
                rearNode = nullptr;
            }
        } else {
```

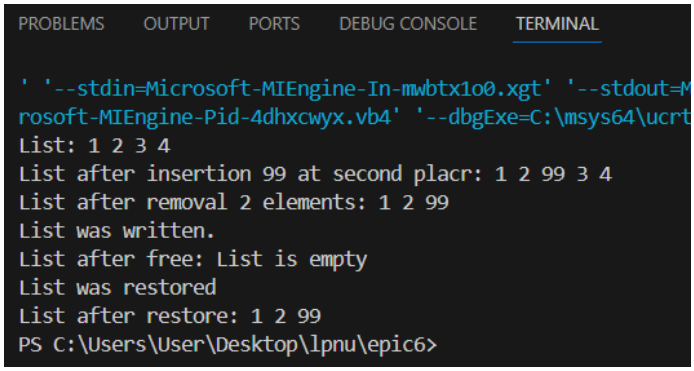
```
            cout << "Черга порожня. Неможливо видалити елемент." << endl;
        }
    }

    int getSize() const {
        return size();
    }

    void printQueue() const {
        if (isEmpty()) {
            cout << "Черга порожня." << endl;
            return;
        }

        Node* current = frontNode;
        cout << "Елементи черги: ";
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }

    ~Queue() {
        while (!isEmpty()) {
            dequeue();
        }
    }
};

int main() {
    Queue myQueue;

    myQueue.enqueue(1);
    myQueue.enqueue(2);
    myQueue.enqueue(3);

    cout << "Розмір черги після додавання елементів: " << myQueue.getSize() <<
endl;
    myQueue.printQueue();

    myQueue.dequeue();
    cout << "Розмір черги після видалення елементу: " << myQueue.getSize() <<
endl;
    myQueue.printQueue();

    return 0;
}
```

Планований час: 40 хв. Фактичний: 40 хв.

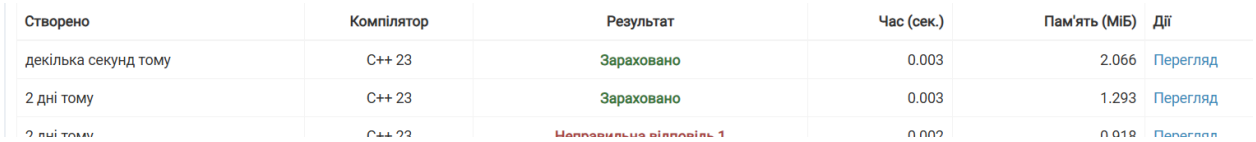
https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/479/files#diff-c8c882bf951b54a93ca245f78dd89ba5c9e44061b300ed7e0ed9bbeb2f69196a

4) *Результати виконання завдань:*

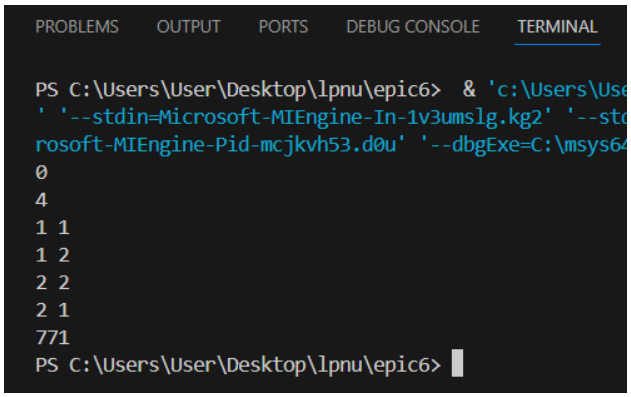
Завдання №1 - VNS Lab 10 - Task 1-10

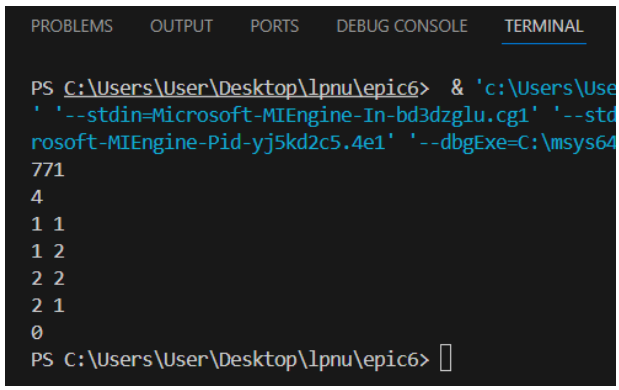


Завдання №2 - Algotester Lab 5



Створено	Компілятор	Результат	Час (сек.)	Пам'ять (МіБ)	Дії
декілька секунд тому	C++ 23	Зараховано	0.003	2.066	Перегляд
2 дні тому	C++ 23	Зараховано	0.003	1.293	Перегляд





Завдання №3 - Algotester Lab 7-8

```
PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE  TERMINAL

rosoft-MIEngine-Pid-jjbnw1dm.ncm' '--dbgExe=C:\msys
11

size
0
insert 5
insert 4
print
4 5
insert 5
print
4 5
insert 1
print
1 4 5
contains 5
Yes
contains 0
No
size
3
PS C:\Users\User\Desktop\lpnu\epic6>
```

Створено	Компілятор	Результат	Час (сек.)	Пам'ять (МіБ)	Дії
2 дні тому	C++ 23	Зараховано	0.008	1.316	Перегляд

Завдання №4 - Class Practice Work - Task 1 - Реверс списку (Reverse list)

```
PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE  TERMINAL

PS C:\Users\User\Desktop\lpnu\epic6>  & 'c:\Users\Use
' '--stdin=Microsoft-MIEngine-In-nqxndrrq.rz2' '--std
rosoft-MIEngine-Pid-2ktew5lt.a3a' '--dbgExe=C:\msys64
Вихідний список: 1 2 3 4
Обернений список: 4 3 2 1
PS C:\Users\User\Desktop\lpnu\epic6>
```

Завдання №5 - Class Practice Work - Task 2 - Порівняння списків

```
44  int main() {
45      Node* head1 = nullptr;
46      pushToEnd(head1, 1);
47      pushToEnd(head1, 2);
48      pushToEnd(head1, 3);
49
50      Node* head2 = nullptr;
51      push(head2, 3);
52      push(head2, 2);
53      push(head2, 1);
54
```

```
PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE  TERMI

PS C:\Users\User\Desktop\lpnu\epic6>  & 'c:\Use
' '--stdin=Microsoft-MIEngine-In-t0tnapil.1ij'
rosoft-MIEngine-Pid-sl0mqm2x.hoq' '--dbgExe=C:\
Списки однакові
PS C:\Users\User\Desktop\lpnu\epic6>
```

Завдання №6 - Class Practice Work - Task 3 - Додавання великих чисел

```
51  int main() {
52
53      Node* n1 = new Node(9);
54      n1->next = new Node(7);
55      n1->next->next = new Node(3);
56
57      Node* n2 = new Node(6);
58      n2->next = new Node(4);
59      n2->next->next = new Node(8);
60
61      Node* result = add(n1, n2);
62
```

```
PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE  T

PS C:\Users\User\Desktop\lpnu\epic6>  & 'c:\
' '--stdin=Microsoft-MIEngine-In-voxa5wye.hi
rosoft-MIEngine-Pid-3wqmeoid.uge' '--dbgExe=
Результат додавання: 5 2 2 1
PS C:\Users\User\Desktop\lpnu\epic6>
```

Завдання №7 - Class Practice Work - Task 4 - Віддзеркалення дерева

```
PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE  TERMINAL

PS C:\Users\User\Desktop\lpnu\epic6>  & 'c:\Users\U
' '--stdin=Microsoft-MIEngine-In-ucvbktvd.xur' '--s
rosoft-MIEngine-Pid-ryw52miu.dbv' '--dbgExe=C:\msys
Original tree: 4 2 5 1 6 3 7
Mirrored tree: 7 3 6 1 5 2 4
PS C:\Users\User\Desktop\lpnu\epic6>
```

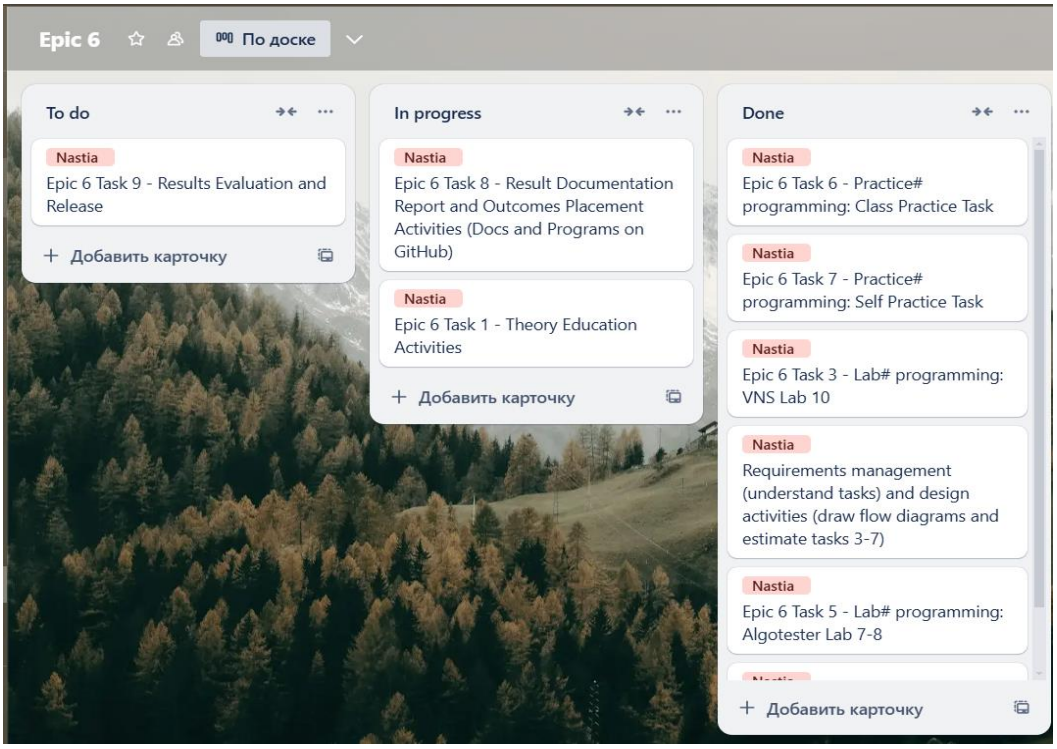
Завдання №8 - Class Practice Work - Task 5 - Записати кожному батьківському вузлу суму підвузлів

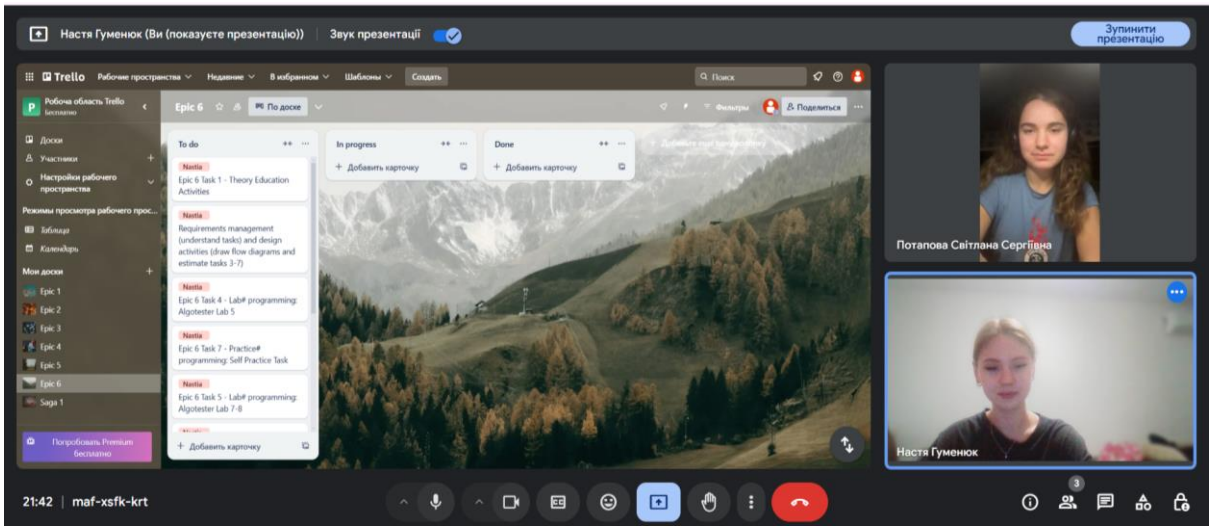
```
PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE  TERMINAL

PS C:\Users\User\Desktop\lpnu\epic6>  & 'c:\Users\User\.v
' '--stdin=Microsoft-MIEngine-In-13oz1scu.2bh' '--stdout=
rosoft-MIEngine-Pid-blli3arx.lxo' '--dbgExe=C:\msys64\ucr
Original tree: 1 2 4 5 3
Sum tree: 15 11 4 5 3
PS C:\Users\User\Desktop\lpnu\epic6>
```

Завдання №9 - Self Practice Work – Черга

```
PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE  TERMINAL

PS C:\Users\User\Desktop\lpnu\epic6>  & 'c:\Users\Us
' '--stdin=Microsoft-MIEngine-In-2ba5vtiy.yv0' '--st
rosoft-MIEngine-Pid-fntqk1jz.pjw' '--dbgExe=C:\msys6
Розмір черги після додавання елементів: 3
Елементи черги: 1 2 3
Розмір черги після видалення елементу: 2
Елементи черги: 2 3
PS C:\Users\User\Desktop\lpnu\epic6>
```

5) Кооперація з командою:





Висновок: Виконуючи 6 епік, я ознайомилася з основними принципами роботи з динамічними структурами даних у C++, зокрема стеком, чергою, зв'язними списками та деревами. Вивчила їх властивості та основні операції, такі як додавання, видалення елементів. Опанувала алгоритми для маніпуляцій з цими структурами, включаючи ітеративні та рекурсивні підходи. Особливу увагу приділила принципам виділення пам'яті для динамічних структур, розумінню їх обробки, а також роботі з переповненням стеку і обробці подій через чергу. Дослідження складніших типів дерев, таких як бінарні, AVL та червоно-чорні, допомогло зрозуміти їх застосування та переваги в реальних задачах. Виконання цих завдань поглибило розуміння принципів роботи з динамічною пам'яттю і покращило навички у використанні алгоритмів для ефективної обробки даних.