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:*** «Мови та парадигми програмування»

до:

ВНС Лабораторної Роботи № 10

Алготестер Лабораторної Роботи № 5

Алготестер Лабораторної Роботи № 7-8

Практичних Робіт № 6

**Виконав:**

Студент групи ШІ-14

Кисіль Сергій Дмитрович

# **Тема роботи:**

# Первинне ознайомлення з динамічними структурами даних, такими як стек, черга, списки, бінарні дерева. Спроби реалізації різних видів списків(однозв’язний лінійний та двозв’язний лінійний), ознайомлення з одно- та двозв’язними циклічними списками. Ознайомлення з елементами об’єктно-орієнтованого програмування(класи, об’єкти, методи, конструктори, деструктори і т.д)

# **Мета роботи:**

Ознайомитись з різними видами динамічних структур, зокрема стеком, чергою, різними видами списків(linked lists), бінарними деревами. Спробувати реалізувати однозв’язний та двозв’язний лінійний список, елементи бінарного дерева. Отримати початкові уявлення про елементи ООП: класи, об’єкти, методи. Ознайомитись з конструкторами та деструкторами, отримати поняття про специфікатори доступу, зрозуміти різницю між класами та структурами.

Завдання:

* Epic 6 Task 1 - Theory Education Activities
* Epic 6 Task 2 - Requirements management (understand tasks) and design activities (draw flow diagrams and estimate tasks 3-7)
* Epic 6 Task 3 - Lab# programming: VNS Lab 10
* Epic 6 Task 4 - Lab# programming: Algotester Lab 5
* Epic 6 Task 5 - Lab# programming: Algotester Lab 7-8
* Epic 6 Task 6 - Practice# programming: Class Practice Task
* Epic 6 Task 7 - Practice# programming: Self Practice Task
* Epic 6 Task 8 - Result Documentation Report and Outcomes Placement Activities (Docs and Programs on GitHub)
* Epic 6 Task 9 - Results Evaluation and Release

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

⦁ Тема №1: Списки.

⦁ Тема №2: Дерева

⦁ Тема №3: Стек

⦁ Тема №4: Черга

1. Індивідуальний план опрацювання теорії:

Тема №1: Списки

* + Джерела Інформації:

<https://www.youtube.com/watch?v=HKfj0l7ndbc&list=PL43pGnjiVwgS5njI0HMGnqSH18tSSuLz_&index=4>

<https://www.youtube.com/watch?v=RNMIDj62o_o&list=PL43pGnjiVwgS5njI0HMGnqSH18tSSuLz_&index=5>

<https://www.youtube.com/watch?v=-StYr9wILqo&list=PL43pGnjiVwgS5njI0HMGnqSH18tSSuLz_&index=9>

Team meats

Одногрупники та однопотоківці

* + Що опрацьовано:

Ознайомлено з теорією

Використано у практичних та лабораторних роботах

* + Статус: Ознайомлений/ Ознайомлений частково / Не ознайомлений
  + Початок опрацювання теми: 01.11.2023
  + Завершення опрацювання теми: 29.11.2023

Тема №2: Дерева

* + Джерела Інформації:

Team meats

<https://www.youtube.com/watch?v=UHxtjVsOTHc>

<https://www.youtube.com/watch?v=_IhTp8q0Mm0>

(також кожну тему опрацьовував на <https://vns.lpnu.ua/pluginfile.php?file=%2F1116827%2Fmod_resource%2Fcontent%2F1%2Fkonspekt_lekcyi_ukr_1.pdf> )

* + Що опрацьовано:

Ознайомлено з теорією

Використано у практичних та лабораторних роботах

* + Статус: Ознайомлений/ Ознайомлений частково / Не ознайомлений
  + Початок опрацювання теми: 01.11.2023
  + Завершення опрацювання теми: 29.11.2023

Тема №3: Стек

* + Джерела Інформації:

<https://www.youtube.com/watch?v=GBST5uQ_yos&list=PL43pGnjiVwgS5njI0HMGnqSH18tSSuLz_&index=6>

Team meats

Одногрупники

* + Що опрацьовано:

Ознайомлено з теорією

Використано у практичних та лабораторних роботах

Статус: Ознайомлений/ Ознайомлений частково / Не ознайомлений

* + Початок опрацювання теми: 01.11.2023
  + Завершення опрацювання теми: 29.11.2023

Тема №4: Черга

* + Джерела Інформації:

<https://www.youtube.com/watch?v=jaK4pn1jXTo&list=PL43pGnjiVwgS5njI0HMGnqSH18tSSuLz_&index=7>

Одногрупники

* + Що опрацьовано:

Ознайомлено з теорією

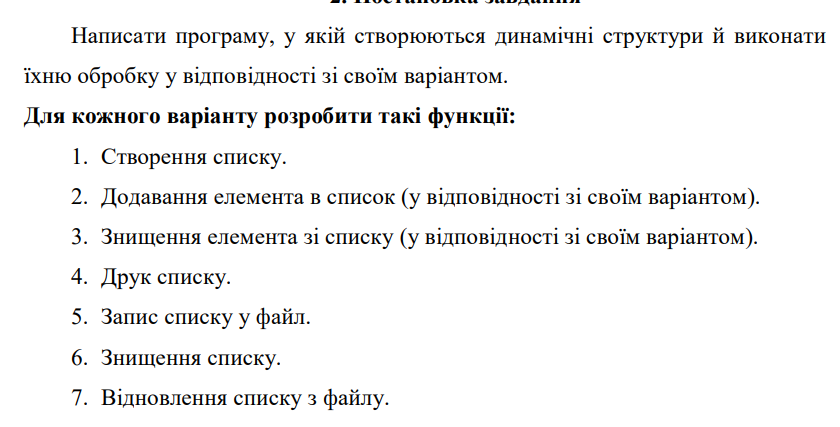
Використано у практичних та лабораторних роботах

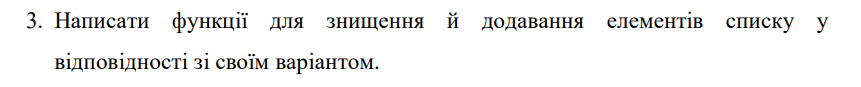
Статус: Ознайомлений/ Ознайомлений частково / Не ознайомлений

* + Початок опрацювання теми: 01.11.2023
  + Звершення опрацювання теми: 29.11.2023

# **Виконання роботи:**

## **1. Опрацювання завдання та вимог до програм та середовища:**

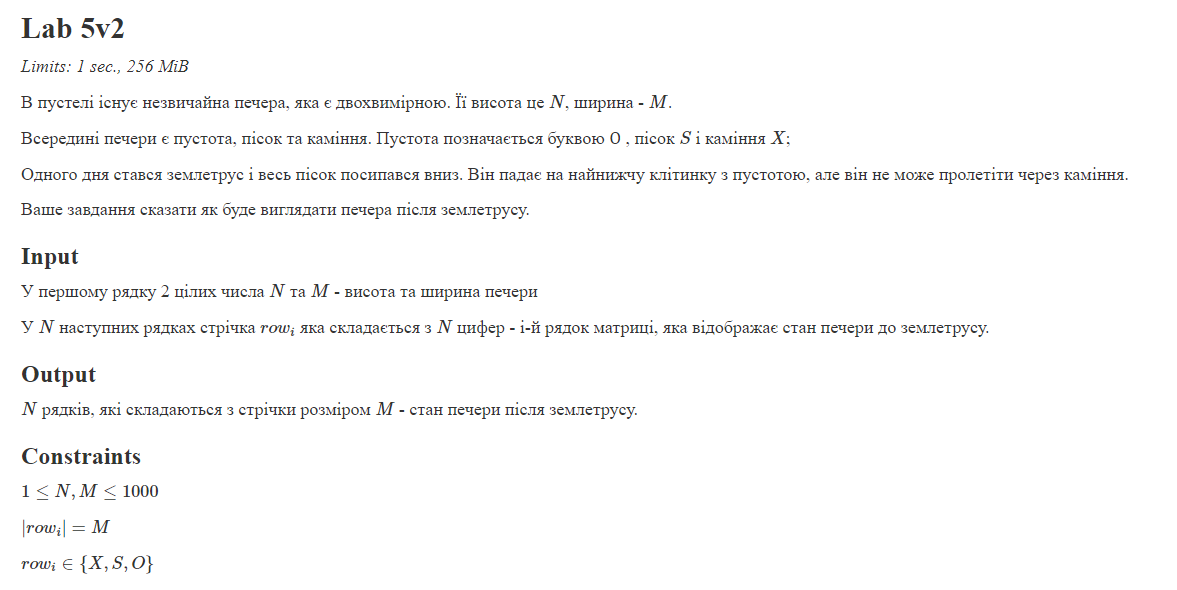
* Завдання №1 Лаб# Програмування: VNS Lab10 (Завдання 1)
* Варіант завдання: 3
* Деталі завдання:



Малюнок(1) - умова завдання 1.

Завдання №2 Лаб# Програмування: Algotester Lab 5

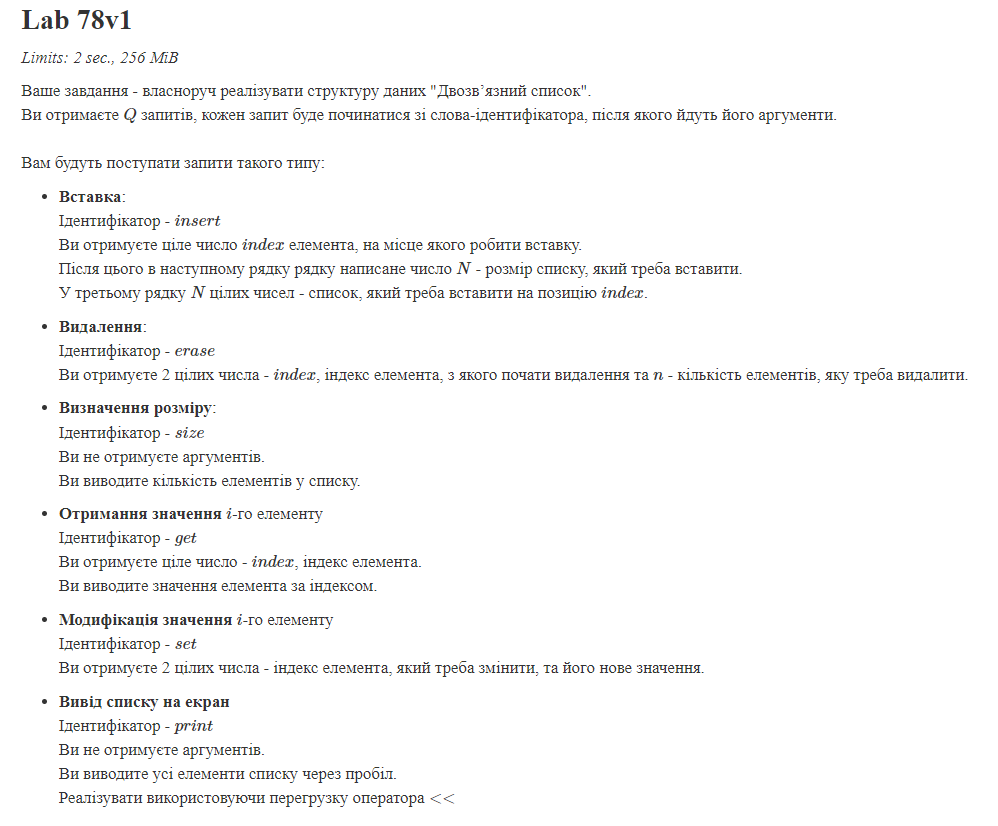
Варіант завдання: 2

* Деталі завдання:
* 

Малюнок(2) - умова завдання 2.

Завдання №3 Лаб# Програмування: Algotester Lab 78

Варіант завдання: 1

* Деталі завдання:

Малюнок(3) - умова завдання 3.

Завдання №4 Лаб# Програмування: Class Practice Task 1

Варіант завдання (відсутній)

* Деталі завдання:

## **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.

Завдання №5 Лаб# Програмування: Class Practice Task 2

Варіант завдання (відсутній)

* Деталі завдання:

## **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.

Завдання №6 Практична# Програмування: Class Practice Task 3

Варіант завдання (відсутній)

* Деталі завдання:

## **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.

Завдання №7 Практична# Програмування: Class Practice Task 4

Варіант завдання (відсутній)

* Деталі завдання:

## **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.

Завдання №8 Лаб# Програмування: Class Practice Task 5

Варіант завдання (відсутній)

* Деталі завдання:

## **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 розумі

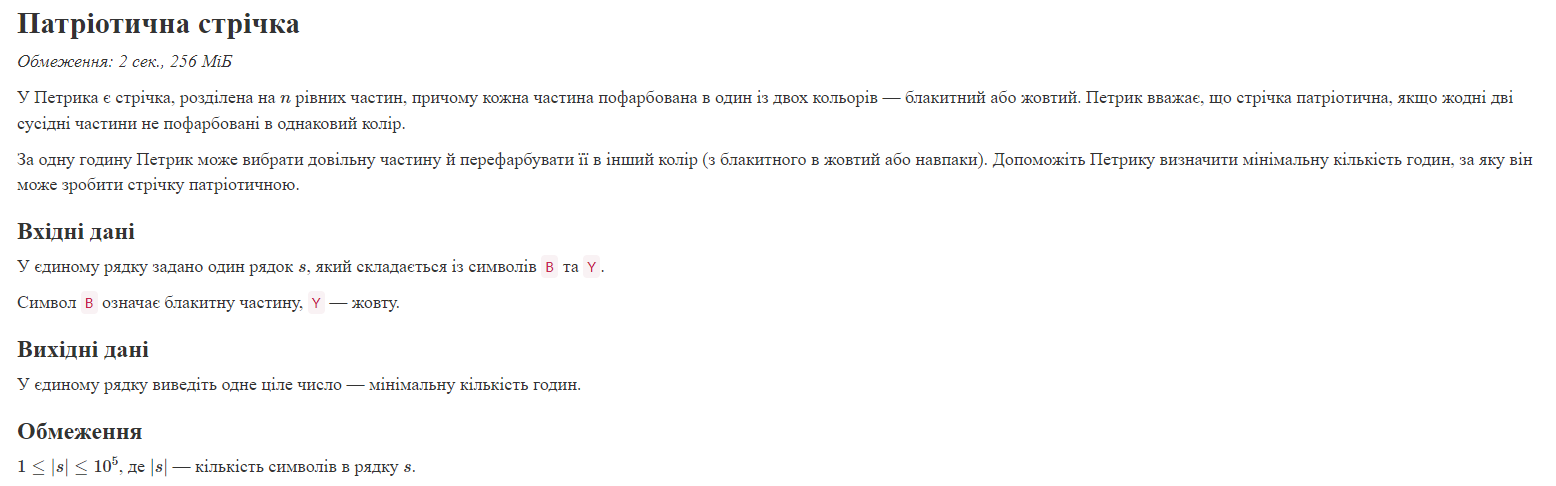
ння структури бінарного дерева. Це один з багатьох методів роботи з бінарними деревами.

***Розвиток алгоритмічне мислення:*** Це завдання розвиває алгоритмічне мислення. Прохід всіх вузлів дерева демонструє розгортання рекурсивного виклику.

Завдання №9 Практична# Програмування: Self Practice Task

Варіант завдання (відсутній)

* Деталі завдання:



Малюнок(4) - умова завдання 9.

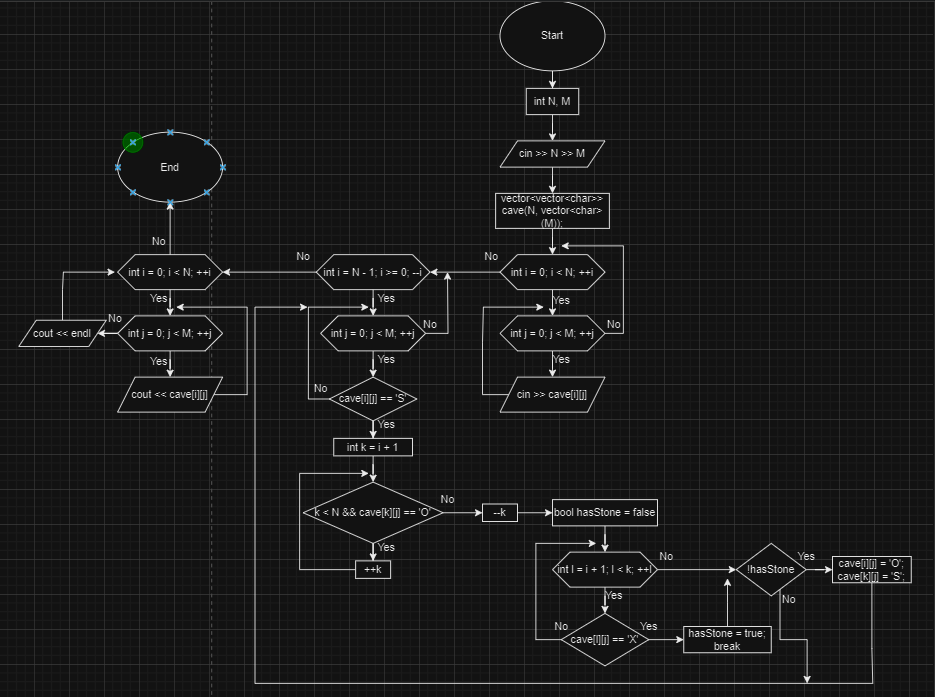
## **2. Дизайн та планована оцінка часу виконання завдань:**

Програма №1 VNS Lab 10

* Планований час на реалізацію: 70 хвилин.

Програма №2 Лаб# Програмування: Algotester Lab 5

* Планований час на реалізацію: 400 хвилин.



Малюнок(5) - блок-схема до Програми № 2.

Програма №3: Лаб# Програмування: Algotester Lab 78

* Планований час на реалізацію 200 хвилин.

Програма №4: Лаб# Програмування: Class Practice Task 1

* Планований час на реалізацію 1.5 години.

Програма №5: Лаб# Програмування: Class Practice Task 2

* Планований час на реалізацію 140 хвилин.

Програма №6: Практична# Програмування: Class Practice Task 3

* Планований час на реалізацію 80 хвилин.

Програма №7: Практична# Програмування: Class Practice Task 4

* Планований час на реалізацію 20 хвилин.

Програма №8: Лаб# Програмування: Class Practice Task 5

* Планований час на реалізацію 100 хвилин.

Програма №9: Практична# Програмування: Self Practice Task

* Планований час на реалізацію 30 хвилин.

## **3. Код програм з посиланням на зовнішні ресурси:**

Завдання №1: Лаб# Програмування: VNS Lab 10

#include <iostream>

#include <fstream>

// Структура, що представляє елемент списку

struct Node {

int data; // ключове поле типу int

Node\* next; // вказівник на наступний елемент списку

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

};

// Клас, що представляє однонаправлений список

class LinkedList {

private:

Node\* head; // вказівник на перший елемент списку

public:

LinkedList() : head(nullptr) {}

// Функція для додавання елемента в кінець списку

void append(int val) {

Node\* newNode = new Node(val);

if (!head) {

head = newNode;

} else {

Node\* current = head;

while (current->next) {

current = current->next;

}

current->next = newNode;

}

}

// Функція для друку списку

void print() {

if (!head) {

std::cout << "List is empty" << std::endl;

return;

}

Node\* current = head;

while (current) {

std::cout << current->data << " ";

current = current->next;

}

std::cout << std::endl;

}

// Функція для знищення K елементів, починаючи з заданого номера

void deleteElements(int startIdx, int K) {

if (!head) {

return;

}

if (startIdx == 0) {

while (K > 0 && head) {

Node\* temp = head;

head = head->next;

delete temp;

K--;

}

} else {

Node\* current = head;

for (int i = 0; i < startIdx - 1 && current; i++) {

current = current->next;

}

if (!current || !current->next) {

return;

}

Node\* prev = current;

current = current->next;

while (K > 0 && current) {

prev->next = current->next;

delete current;

current = prev->next;

K--;

}

}

}

// Функція для додавання елемента перед елементом із заданим ключем

void insertBefore(int key, int val) {

Node\* newNode = new Node(val);

if (!head) {

head = newNode;

return;

}

if (head->data == key) {

newNode->next = head;

head = newNode;

return;

}

Node\* current = head;

while (current->next) {

if (current->next->data == key) {

newNode->next = current->next;

current->next = newNode;

return;

}

current = current->next;

}

}

// Функція для запису списку у файл

void saveToFile(const std::string& filename) {

std::ofstream file(filename);

if (!file.is\_open()) {

std::cerr << "Failed to open file" << std::endl;

return;

}

Node\* current = head;

while (current) {

file << current->data << " ";

current = current->next;

}

file.close();

}

// Функція для знищення списку

void destroy() {

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

};

int main() {

LinkedList list;

// Додавання елементів

list.append(1);

list.append(2);

list.append(3);

list.append(4);

list.append(5);

// Друк списку

std::cout << "List:" << std::endl;

list.print();

// Видалення K елементів, починаючи з заданого номера

int startIdx = 2; // починаємо з третього елемента (індекс 2)

int K = 2; // видаляємо 2 елементи

list.deleteElements(startIdx, K);

std::cout << "List after deleting:" << std::endl;

list.print();

// Додавання елемента перед елементом із заданим ключем

int key = 1; // шукаємо елемент з ключем 1

int val = 6; // додаємо 6 перед елементом з ключем 1

list.insertBefore(key, val);

std::cout << "List after adding:" << std::endl;

list.print();

// Запис списку у файл

list.saveToFile("list.txt");

// Знищення списку

list.destroy();

// Друк порожнього списку

std::cout << "List after removing:" << std::endl;

list.print();

// Відновлення списку з файлу

LinkedList restoredList;

std::ifstream file("list.txt");

int value;

while (file >> value) {

restoredList.append(value);

}

std::cout << "Rebuild list:" << std::endl;

restoredList.print();

// Знищення відновленого списку

restoredList.destroy();

return 0;

}

Завдання №2: Лаб# Програмування: Algotester Lab 5

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

int N, M;

cin >> N >> M;

vector<vector<char>> cave(N, vector<char>(M));

// Зчитування стану печери

for (int i = 0; i < N; ++i) {

for (int j = 0; j < M; ++j) {

cin >> cave[i][j];

}

}

// Проходимо через кожен рядок печери знизу вгору

for (int i = N - 1; i >= 0; --i) {

for (int j = 0; j < M; ++j) {

if (cave[i][j] == 'S') {

// Знаходимо найнижчий блок O

int k = i + 1;

while (k < N && cave[k][j] == 'O') {

++k;

}

--k;

// Перевіряємо, чи є камінь X на шляху до блока O

bool hasStone = false;

for (int l = i + 1; l < k; ++l) {

if (cave[l][j] == 'X') {

hasStone = true;

break;

}

}

// Якщо немає каменя, переміщуємо пісок S вниз

if (!hasStone) {

cave[i][j] = 'O';

cave[k][j] = 'S';

}

}

}

}

// Виводимо стан печери після землетрусу

for (int i = 0; i < N; ++i) {

for (int j = 0; j < M; ++j) {

cout << cave[i][j];

}

cout << endl;

}

return 0;

}

Завдання №3: Лаб# Програмування: Algotester Lab 78

#include <iostream>

#include <string>

#include <vector>

using namespace std;

class DoubleLinkedList {

private:

struct Node {

int data;

Node\* next;

Node\* prev;

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

};

Node\* head;

Node\* tail;

int listSize;

public:

DoubleLinkedList() : head(nullptr), tail(nullptr), listSize(0) {}

void insert(int index, int N, const vector<int>& Ni) {

if (index > listSize || index < 0 || N <= 0)

return;

Node\* current = head;

for (int i = 0; i < index; i++)

current = current->next;

Node\* prevNode = current ? current->prev : nullptr;

for (int i = 0; i < N; i++) {

Node\* newNode = new Node(Ni[i]);

newNode->prev = prevNode;

if (prevNode)

prevNode->next = newNode;

else

head = newNode;

newNode->next = current;

if (current)

current->prev = newNode;

else

tail = newNode;

prevNode = newNode;

listSize++;

}

}

void erase(int index, int N) {

if (index < 0 || index >= listSize || N <= 0)

return;

Node\* current = head;

for (int i = 0; i < index; i++)

current = current->next;

Node\* prevNode = current ? current->prev : nullptr;

Node\* nextNode = current ? current->next : nullptr;

for (int i = 0; i < N && current != nullptr; i++) {

delete current;

current = nextNode;

listSize--;

if (nextNode)

nextNode = nextNode->next;

if (prevNode)

prevNode->next = nextNode;

else

head = nextNode;

if (nextNode)

nextNode->prev = prevNode;

else

tail = prevNode;

}

}

int size() const {

return listSize;

}

int get(int index) const {

if (index < 0 || index >= listSize) {

cout << "Invalid index" << endl;

return -1; // Ви можете обрати інший значок помилки за необхідності

}

Node\* current = head;

for (int i = 0; i < index; i++)

current = current->next;

return current->data;

}

void set(int index, int num) {

if (index < 0 || index >= listSize)

return;

Node\* current = head;

for (int i = 0; i < index; i++)

current = current->next;

current->data = num;

}

friend ostream& operator<<(ostream& os, const DoubleLinkedList& list) {

Node\* current = list.head;

while (current != nullptr) {

os << current->data << " ";

current = current->next;

}

return os;

}

~DoubleLinkedList() {

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

};

int main() {

DoubleLinkedList list;

int q;

cin >> q;

vector<string> identifiers(q);

for (int i = 0; i < q; i++) {

cin >> identifiers[i];

if (identifiers[i] == "insert") {

int index, N;

cin >> index >> N;

vector<int> NI(N);

for (int j = 0; j < N; j++) {

cin >> NI[j];

}

list.insert(index, N, NI);

} else if (identifiers[i] == "erase") {

int index, N;

cin >> index >> N;

list.erase(index, N);

} else if (identifiers[i] == "size") {

cout << list.size() << endl;

} else if (identifiers[i] == "get") {

int index;

cin >> index;

int value = list.get(index);

if (value != -1)

cout << value << endl;

} else if (identifiers[i] == "set") {

int index, num;

cin >> index >> num;

list.set(index, num);

} else {

cout << list << endl;

}

}

return 0;

}

Завдання №4: Лаб# Програмування: Class Practice Task 1

#include <iostream>

class Node {

public:

int data;

Node\* next;

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

};

Node\* reverse(Node\* head) {

Node\* prev = nullptr;

Node\* current = head;

Node\* next = nullptr;

while (current != nullptr) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

return prev; // новий початок списку

}

void printList(Node\* head) {

Node\* current = head;

while (current != nullptr) {

std::cout << current->data << " ";

current = current->next;

}

std::cout << std::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);

head->next->next->next->next = new Node(5);

std::cout << "Оригінальний список: ";

printList(head);

head = reverse(head);

std::cout << "Обернений список: ";

printList(head);

return 0;

}

Завдання №5: Лаб# Програмування: Class Practice Task 2

#include <iostream>

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; // Знайдено невідповідність даних, повертаємо false

}

h1 = h1->next;

h2 = h2->next;

}

// Один із списків закінчився раніше іншого, тоді вони різні

if (h1 != nullptr || h2 != nullptr) {

return false;

}

// Обидва списки пройшлися і дані відповідають один одному

return true;

}

int main() {

// Приклади списків для порівняння

Node\* list1 = new Node(1);

list1->next = new Node(2);

list1->next->next = new Node(3);

Node\* list2 = new Node(1);

list2->next = new Node(2);

list2->next->next = new Node(3);

Node\* list3 = new Node(1);

list3->next = new Node(2);

// Порівняння списків

if (compare(list1, list2)) {

std::cout << "list1 і list2 однакові" << std::endl;

} else {

std::cout << "list1 і list2 різні" << std::endl;

}

if (compare(list1, list3)) {

std::cout << "list1 і list3 однакові" << std::endl;

} else {

std::cout << "list1 і list3 різні" << std::endl;

}

// Звільнення пам'яті

while (list1 != nullptr) {

Node\* temp = list1;

list1 = list1->next;

delete temp;

}

while (list2 != nullptr) {

Node\* temp = list2;

list2 = list2->next;

delete temp;

}

while (list3 != nullptr) {

Node\* temp = list3;

list3 = list3->next;

delete temp;

}

return 0;

}

Завдання №6: Практична# Програмування: Class Practice Task 3

#include <iostream>

class Node {

public:

int data;

Node\* next;

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

};

Node\* add(Node\* n1, Node\* n2) {

Node\* result = nullptr;

Node\* current = nullptr;

int carry = 0; // перенос для додавання

while (n1 != nullptr || n2 != nullptr) {

int sum = carry;

if (n1 != nullptr) {

sum += n1->data;

n1 = n1->next;

}

if (n2 != nullptr) {

sum += n2->data;

n2 = n2->next;

}

carry = sum / 10; // визначаємо перенос

sum = sum % 10; // визначаємо поточну цифру результуючого числа

Node\* newNode = new Node(sum);

if (result == nullptr) {

result = newNode;

current = result;

} else {

current->next = newNode;

current = current->next;

}

}

if (carry > 0) {

Node\* newNode = new Node(carry);

current->next = newNode;

}

return result;

}

void printList(Node\* head) {

Node\* current = head;

while (current != nullptr) {

std::cout << current->data;

current = current->next;

}

std::cout << std::endl;

}

int main() {

Node\* num1 = new Node(3);

num1->next = new Node(7);

num1->next->next = new Node(9);

Node\* num2 = new Node(4);

num2->next = new Node(2);

num2->next->next = new Node(5);

std::cout << "Число 1: ";

printList(num1);

std::cout << "Число 2: ";

printList(num2);

Node\* result = add(num1, num2);

std::cout << "Сума: ";

printList(result);

return 0;

}

Завдання №7: Практична# Програмування: Class Practice Task 4

#include <iostream>

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int x) : val(x), left(NULL), right(NULL) {}

};

TreeNode\* create\_mirror\_flip(TreeNode\* root) {

if (root == NULL) {

return NULL;

}

// Створюємо новий вузол зі значенням root->val

TreeNode\* newRoot = new TreeNode(root->val);

// Рекурсивно віддзеркалюємо праву піддерево і присвоюємо його лівій гілці нового вузла

newRoot->left = create\_mirror\_flip(root->right);

// Рекурсивно віддзеркалюємо ліве піддерево і присвоюємо його правій гілці нового вузла

newRoot->right = create\_mirror\_flip(root->left);

return newRoot;

}

// Функція для виведення дерева (інфіксний обхід)

void inorderTraversal(TreeNode\* root) {

if (root == NULL) {

return;

}

inorderTraversal(root->left);

std::cout << root->val << " ";

inorderTraversal(root->right);

}

int main() {

// Приклад вхідного дерева

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);

std::cout << "Вхідне дерево (інфіксний обхід): ";

inorderTraversal(root);

std::cout << std::endl;

TreeNode\* mirroredRoot = create\_mirror\_flip(root);

std::cout << "Дзеркальне дерево (інфіксний обхід): ";

inorderTraversal(mirroredRoot);

std::cout << std::endl;

// Звільнення пам'яті

// Видалення дерева за допомогою рекурсивної функції

return 0;

}

Завдання №8: Лаб# Програмування: Class Practice Task 5

#include <iostream>

#include <stack>

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int x) : val(x), left(NULL), right(NULL) {}

};

void tree\_sum(TreeNode\* root) {

if (root == NULL) {

return;

}

std::stack<TreeNode\*> nodes;

TreeNode\* current = root;

TreeNode\* prev = NULL;

while (current != NULL || !nodes.empty()) {

while (current != NULL) {

nodes.push(current);

current = current->right;

}

current = nodes.top();

nodes.pop();

if (current->left == NULL && current->right == NULL) {

// Якщо це листок, нічого не змінюємо

} else {

int left\_sum = (current->left != NULL) ? current->left->val : 0;

int right\_sum = (current->right != NULL) ? current->right->val : 0;

current->val = left\_sum + right\_sum;

}

prev = current;

current = current->left;

}

}

// Функція для виведення дерева (інфіксний обхід)

void inorderTraversal(TreeNode\* root) {

if (root == NULL) {

return;

}

inorderTraversal(root->left);

std::cout << root->val << " ";

inorderTraversal(root->right);

}

int main() {

// Приклад дерева

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);

std::cout << "Вхідне дерево (інфіксний обхід): ";

inorderTraversal(root);

std::cout << std::endl;

tree\_sum(root);

std::cout << "Дерево зі сумами підвузлів: ";

inorderTraversal(root);

std::cout << std::endl;

// Звільнення пам'яті

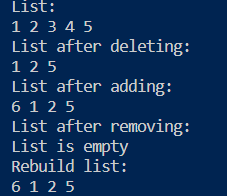
return 0;

}

Завдання №9: Практична# Програмування: Self Practice Task

## **4. Результати виконання завдань, тестування та фактично затрачений час:**

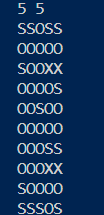
Завдання №1: Лаб# Програмування: VNS Lab 10



Малюнок(6) - результат виконання завдання 1.

Затрачений час: 200 хв.

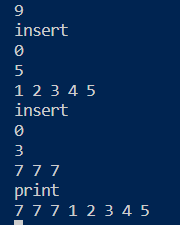
Завдання №2: Лаб# Програмування: Algotester Lab 5



Малюнок(7) - результат виконання завдання 2.

Затрачений час: 5 годин.

Завдання №3: Лаб# Програмування: Algotester Lab 78



Малюнок(8) - результат виконання завдання 3.

Затрачений час: 3 години.

Завдання №4: Лаб# Програмування: Class Practice Task 1



Малюнок(9) - результат виконання завдання 4.

Затрачений час: 2 години.

Завдання №5: Лаб# Програмування: Class Practice Task 2



Малюнок(10) - результат виконання завдання 5.

Затрачений час: 360 хвилин.

Завдання №6: Практична# Програмування: Class Practice Task 3



Малюнок(11) - результат виконання завдання 6.

Затрачений час: 300 хвилин.

Завдання №7: Практична# Програмування: Class Practice Task 4



Малюнок(12) - умова завдання 7.

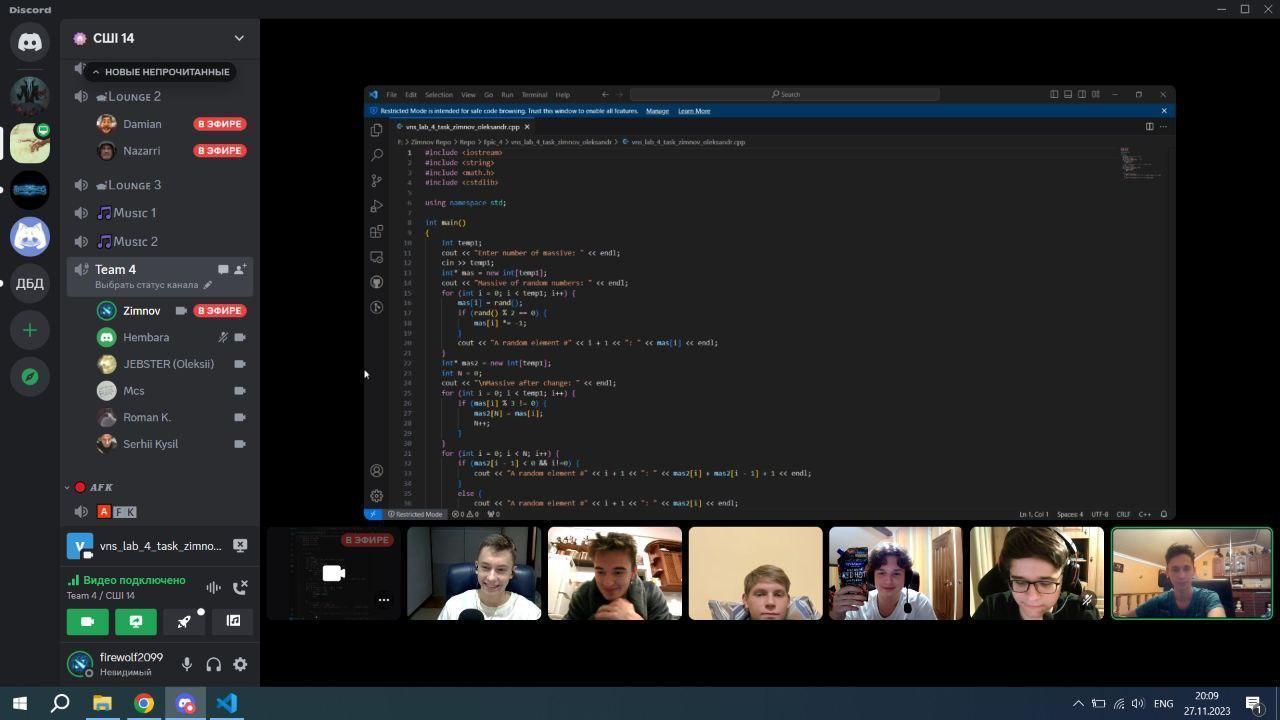
Затрачений час: 1000 хвилин.

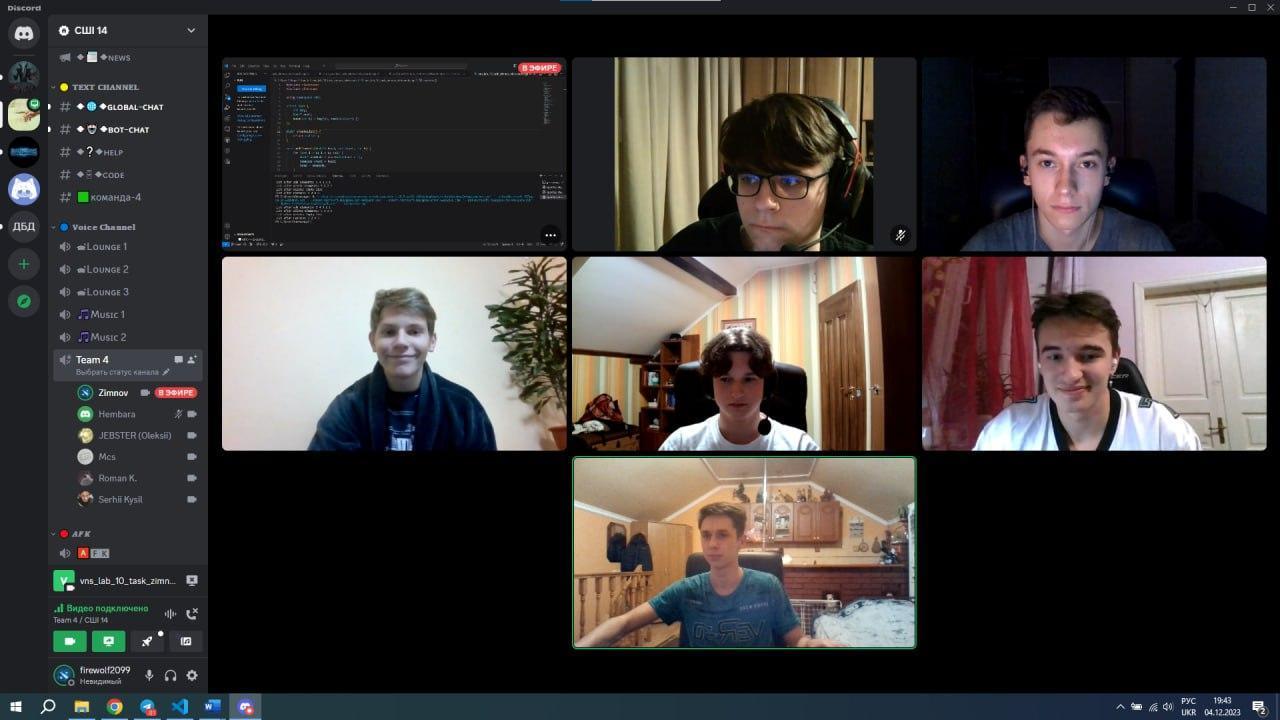
Завдання №8: Лаб# Програмування: Class Practice Task 5

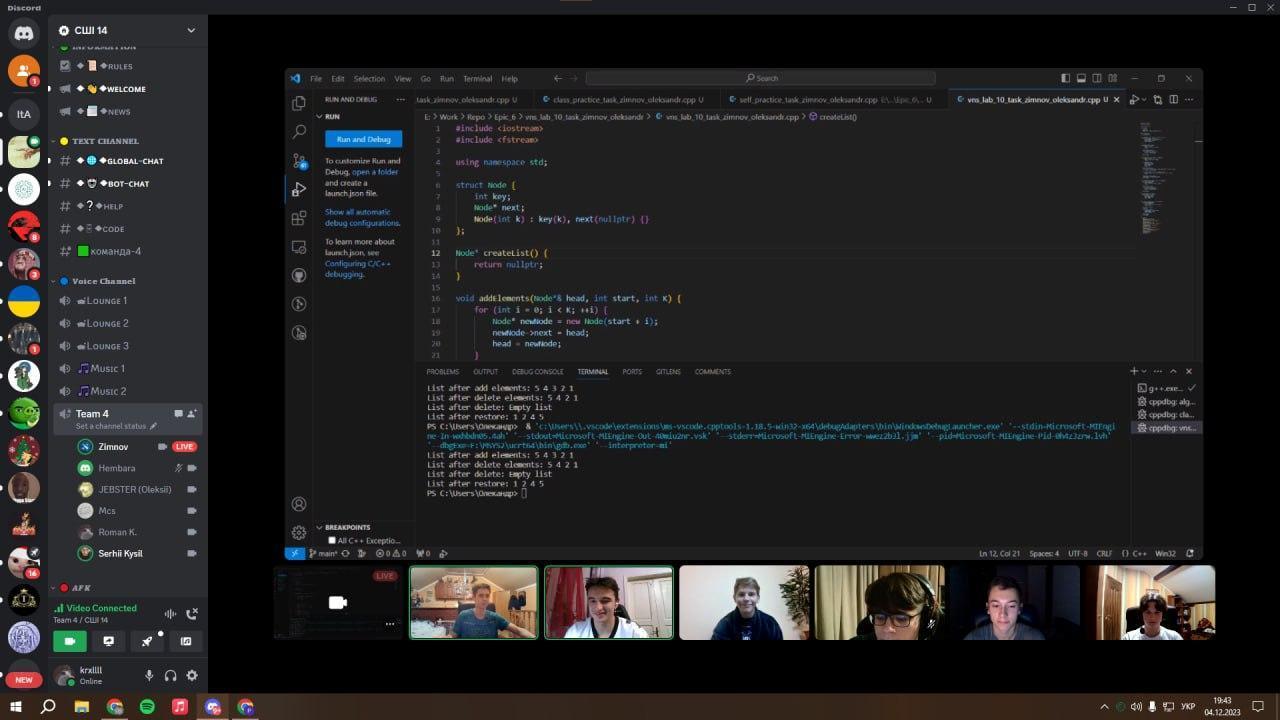


Малюнок(13) - результат виконання завдання 8.

Затрачений час: 600 хвилин.







Малюнки(14-16) - скріни-підтвердження активності команди.

# **Висновки:**

⦁Ознайомлений зі списками

⦁Ознайомлений з деревами

⦁Ознайомлений зі стеками

⦁Ознайомлений з чергами

⦁Успішно виконав лабораторні та практичні роботи

PR: <https://github.com/artificial-intelligence-department/ai_programming_playground/pull/1081>