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:*** «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

Кроляк Роман Романович

# **Тема роботи:**

Динамічні структури. Алгоритми обробки динамічних структур.

# **Мета роботи:**

# Навчитися працювати з динамічними структурами

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Динамічні структури
* Тема №2: Алгоритми обробки динамічних структур

1. Індивідуальний план опрацювання теорії:

* Тема №1: Динамічні структури
  + Джерела Інформації
    - <https://youtu.be/-25REjF_atI?si=6Jh6BrEObodMyEsv>
    - <https://youtu.be/QLzu2-_QFoE?si=BYq0JTCntJLXttup>
    - <https://youtu.be/Yhw8NbjrSFA?si=rYAbUG0eoa2xvQXZ>
    - <https://youtu.be/ZYvYISxaNL0?si=Rofany4fuJMWEkSY>
    - <https://youtu.be/qBFzNW0ALxQ?si=lmKXJp8iSvrSyxu3>
  + Що опрацьовано:
    - Роботу з чергами, стеками, списками, деревами
  + Статус: Ознайомлений
  + Початок опрацювання теми: 07.12.2023
  + Звершення опрацювання теми: 08.12.2023
* Тема №2: Алгоритми обробки динамічних структур
  + Джерела Інформації:
    - <https://acode.com.ua/urok-207-algorytmy-stl/>
  + Що опрацьовано:
    - Такі операції, як пошук, сортування, вставка, видалення і копіювання елементів контейнера.
  + Статус: Ознайомлений
  + Початок опрацювання теми: 02.12.2023
  + Звершення опрацювання теми: 03.12.2023

# **Виконання роботи:**

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

Завдання №1 VNS Lab 10

* Варіант завдання: 2
* 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 однонаправлений список. Знищити з нього елемент із заданим ключем, додати елемент перед елементом із заданим ключем;

Завдання №2 Algotester Lab 5

* Варіант завдання: 3
* У вас є карта гори розміром N×M.
* Також ви знаєте координати {x,y}, у яких знаходиться вершина гори.
* Ваше завдання - розмалювати карту таким чином, щоб найнижча точка мала число 0, а пік гори мав найбільше число.
* Клітинки які мають суміжну сторону з вершиною мають висоту на один меншу, суміжні з ними і не розфарбовані мають ще на 1 меншу висоту і так далі.

Завдання №3 Algotester Lab 7-8

* Варіант завдання: 3

# 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 value - 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 перевантаження оператора <<

# Завдання №4.1 Class Practice Task 1-3

## 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;

## 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*.

## 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.

Завдання №4.2 Class Practice Task 4-5

## 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;

## 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;

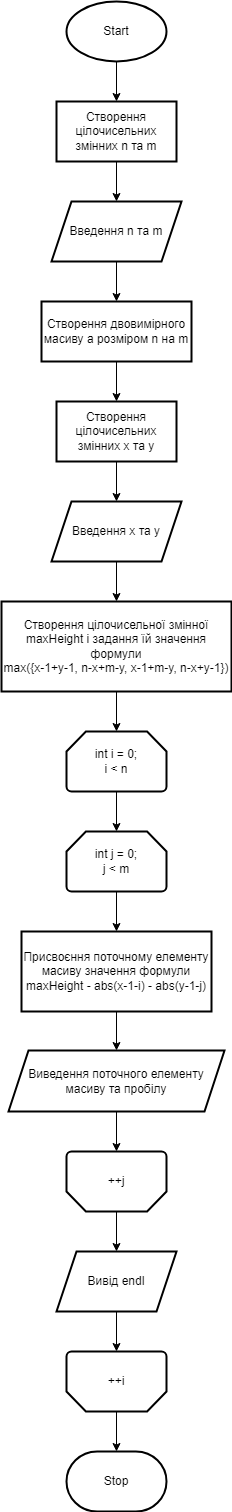
значення змінюються від листків до кореня дерева;

Завдання № 5 - Self Practice Task A  
 [Зуби | Архів | Алготестер](https://algotester.com/uk/ArchiveProblem/DisplayWithEditor/20075)

* Мале Бісеня любить гострити зуби. А Зла Тітонька любить до нього підходити і питатися: «Що, зуби гостриш?». Бісеняті таке не дуже подобається, тому воно придумало робити таке.
* У Малого Бісеняти є n зубів. Кожен зуб має коефіцієнт загостреності ai. Також існує межа загостреності k. Якщо коефіцієнт загостреності певного зуба є більшим чи рівним межі загостреності, то такий зуб вважається загостреним.
* Мале Бісеня хоче наступного разу, коли Зла Тітонька його щось запитає, показати їй якнайбільше загострених зубів, що розташовані поспіль.
* Допоможіть Малому Бісеняті дізнатися, скільки найбільше зубів воно зможе показати.

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

Програма №2 Algotester Lab 5v3

* 

*Рисунок 1 Блок схема до програми №2*

* Планований час на реалізацію: 1 год

## **3. Конфігурація середовища до виконання завдань:**

## Змін у конфігурацію середовища не вносив.

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

Завдання №1 VNS Lab 10

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class List {

public:

// Constructors

List()

: head(nullptr)

{};

[[maybe\_unused]] explicit List(const string& filename)

: head(nullptr)

{

copyFromFile(filename);

};

// Add methods

void pushBack(const int& value){

if (head == nullptr){

head = new Node{value, nullptr};

} else{

Node\* current = head;

while(current->next != nullptr){

current = current->next;

}

current->next = new Node{value, nullptr};

}

};

void insertBefore(const int& value, const int& key){

if (head == nullptr){

head = new Node{value, nullptr};

} else {

Node \*previous = nullptr;

Node \*current = head;

while (current != nullptr) {

if (current->data == key) {

if (previous == nullptr) {

head = new Node{value, current};

} else {

previous->next = new Node{value, current};

}

return;

}

previous = current;

current = current->next;

}

}

};

// Remove methods

void remove(const int& value){

Node\* previous = nullptr;

Node\* current = head;

while(current != nullptr){

if (current->data == value){

if (previous == nullptr){

head = current->next;

} else{

previous->next = current->next;

}

delete current;

return;

}

previous = current;

current = current->next;

}

};

void clear() {

while(head != nullptr){

Node\* current = head;

head = head->next;

delete current;

}

};

// Print method

void print() const {

if (head == nullptr){

cout << "List is empty\n";

} else{

Node\* current = head;

while(current != nullptr){

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

current = current->next;

}

cout << endl;

}

};

// Write to file method

void writeToFile(const string& filename){

if (head == nullptr){

cout << "List is empty\n";

} else{

ofstream file(filename, ios::trunc);

if (file.is\_open()) {

Node \*current = head;

while (current != nullptr) {

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

current = current->next;

}

file << endl;

file.close();

} else cout << "Error! File cannot be opened!" << endl;

}

};

// Copy from file method

void copyFromFile(const string& filename){

clear();

ifstream file(filename);

if (file.is\_open()){

string text;

getline(file, text);

if(text.empty()) return;

head = new Node{int(text[0]) - 48, nullptr};

Node\* current = head;

for (int i = 1; i < text.length(); i++) {

if(text[i] != ' ') {

string s;

for (int j = i; text[j] != ' ' && j < text.length(); ++j) {

s.push\_back(text[j]);

i++;

}

current->next = new Node{stoi(s), nullptr};

current = current->next;

}

}

file.close();

} else cout << "Error! File cannot be opened!" << endl;

}

private:

struct Node{

int data;

Node\* next;

};

Node\* head;

};

int main() {

List list;

for (int i = 0; i < 10; ++i) {

list.pushBack(i);

}

list.print();

cout << "Remove 5\n";

list.remove(5);

list.print();

cout << "Insert 103 before 7\n";

list.insertBefore(103,7);

list.print();

cout << "Write to file, clear and print\n";

list.writeToFile("list.txt");

list.clear();

list.print();

cout << "Copy from file, print\n";

list.copyFromFile("list.txt");

list.print();

list.clear();

return 0;

}

<https://github.com/artificial-intelligence-department/ai_programming_playground/pull/1127/files#diff-d54bb5b8144b1cb5c84dc0c937559d9198d7257e94fbcc2d9e755403a441a684>

Завдання №2 Algotester Lab 5

#include <iostream>

#include <algorithm>

using namespace std;

int main() {

int n, m;

cin >> n >> m;

int a[n][m];

int x, y;

cin >> x >> y;

int maxHeight = max({x-1+y-1, n-x+m-y, x-1+m-y, n-x+y-1});

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

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

a[i][j] = maxHeight - abs(x-1-i) - abs(y-1-j);

cout << a[i][j] << " ";

}

cout << endl;

}

return 0;

}

<https://github.com/artificial-intelligence-department/ai_programming_playground/pull/1127/files#diff-49e161211a13c587bd1a40a1d12e8d9ef1f73645fc328c6df659c0435146dd5a>

Завдання №3 Algotester Lab 7-8

#include <iostream>

using namespace std;

template <typename T>

class Tree {

private:

struct Node {

T data;

Node\* right;

Node\* left;

};

Node\* root;

bool insertNode(Node\*& current, const T& value) {

if (current == nullptr) {

current = new Node{value};

return true;

} else if (value < current->data) {

return insertNode(current->left, value);

} else if (value > current->data) {

return insertNode(current->right, value);

} else {

return false;

}

}

bool containsNode(const Node\* current, const T& value) const {

if (current == nullptr) {

return false;

} else if (value == current->data) {

return true;

} else if (value < current->data) {

return containsNode(current->left, value);

} else {

return containsNode(current->right, value);

}

}

size\_t sizeNode(const Node\* node) const {

return node == nullptr ? 0 : sizeNode(node->left) + 1 + sizeNode(node->right);

}

void printNode(const Node\* node) const {

if (node != nullptr) {

printNode(node->left);

cout << node->data << " ";

printNode(node->right);

}

}

void printNode(const Node\* node, ostream& os) const {

if (node != nullptr) {

printNode(node->left, os);

os << node->data << " ";

printNode(node->right, os);

}

}

public:

Tree()

: root(nullptr) {}

bool insert(const T& value) {

return insertNode(root, value);

}

[[nodiscard]] bool contains(const T& value) const {

return containsNode(root, value);

}

[[nodiscard]] size\_t size() const {

return sizeNode(root);

}

void print() const {

printNode(root);

cout << endl;

}

friend ostream& operator<<(ostream& os, const Tree& tree) {

tree.printNode(tree.root, os);

return os;

}

};

int main() {

Tree<int> tree;

int q, v;

cin >> q;

string s;

cout << endl;

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

cin >> s;

if(s == "insert"){

cin >> v;

tree.insert(v);

}

else if(s == "print"){

cout << tree << endl;

}

else if(s == "contains"){

cin >> v;

tree.contains(v) ? cout << "Yes\n" : cout << "No\n";

}

else if(s == "size"){

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

}

}

return 0;

}

<https://github.com/artificial-intelligence-department/ai_programming_playground/pull/1127/files#diff-7bb3814c048277fd862f9e7082415eb300db0d4ee7b5d2f25a844139d3340485>

# Завдання №4.1 Class Practice Task 1-3

#include <iostream>

#include <ctime>

using namespace std;

class List {

private:

struct Node{

int data;

Node\* next;

};

Node\* head;

void copy(const List& other){

if (other.head == nullptr){

head = nullptr;

} else {

head = new Node{other.head->data, nullptr};

Node \*current = head;

Node \*otherCurrent = other.head;

while (otherCurrent->next != nullptr) {

current->next = new Node{otherCurrent->next->data, nullptr};

current = current->next;

otherCurrent = otherCurrent->next;

}

}

}

public:

// Constructors

List()

: head(nullptr)

{};

List(const List& other)

: head(nullptr)

{

copy(other);

};

// Add methods

void pushBack(const int& value){

if (head == nullptr){

head = new Node{value, nullptr};

} else{

Node\* current = head;

while(current->next != nullptr){

current = current->next;

}

current->next = new Node{value, nullptr};

}

};

// Remove methods

void remove(const int& value){

Node\* previous = nullptr;

Node\* current = head;

while(current != nullptr){

if (current->data == value){

if (previous == nullptr){

head = current->next;

} else{

previous->next = current->next;

}

delete current;

return;

}

previous = current;

current = current->next;

}

};

void clear() {

while(head != nullptr){

Node\* current = head;

head = head->next;

delete current;

}

};

// Print method

void print() const {

if (head == nullptr){

cout << "List is empty\n";

} else{

Node\* current = head;

while(current != nullptr){

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

current = current->next;

}

cout << endl;

}

};

static void print(Node\* node) {

if (node == nullptr){

cout << "List is empty\n";

} else{

Node\* current = node;

while(current != nullptr){

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

current = current->next;

}

cout << endl;

}

};

// Reverse method

Node\* reverse() {

List rev(\*this);

Node\* current = rev.head;

Node\* next\_r;

Node\* prev\_r = nullptr;

while(current != nullptr){

next\_r = current->next;

current->next = prev\_r;

prev\_r = current;

current = next\_r;

}

rev.head = prev\_r;

return rev.head;

};

// Compare method

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

Node\* current1 = h1;

Node\* current2 = h2;

while(current1 != nullptr && current2 != nullptr){

if (current1->data != current2->data){

return false;

}

current1 = current1->next;

current2 = current2->next;

}

return (current1 == nullptr && current2 == nullptr);

};

[[nodiscard]] bool compare(const List& other) const {

return compare(head, other.head);

}

// Big numbers add method

static List add(Node\* h1, Node\* h2){

Node\* current1 = h1;

Node\* current2 = h2;

List sum\_list;

int remainder = 0;

while(current1 != nullptr || current2 != nullptr){

int sum = remainder;

if (current1 != nullptr) {

sum += current1->data;

current1 = current1->next;

}

if (current2 != nullptr) {

sum += current2->data;

current2 = current2->next;

}

remainder = sum / 10;

sum\_list.pushBack(sum%10);

}

if (remainder > 0){

sum\_list.pushBack(remainder);

}

return sum\_list;

};

[[nodiscard]] List add(const List& other) const {

return add(head, other.head);

}

};

int main() {

srand(static\_cast<unsigned>(time(nullptr)));

cout << "Reverse list demo: \n";

List list;

for (int i = 0; i < 10; ++i)

list.pushBack(i);

list.print();

List::print(list.reverse());

cout << "Compare lists demo: \n";

List list1;

List list2;

for (int i = 0; i < 10; ++i)

list1.pushBack(i);

list1.print();

for (int i = 0; i < 20; i+=2)

list2.pushBack(i);

list2.print();

list1.compare(list2) ? cout << "Equal\n" : cout << "Not equal\n";

cout << "Add big numbers demo: \n";

List list3;

for (int i = 0; i < 20; ++i)

list3.pushBack(rand() % 10);

List::print(list3.reverse());

List list4;

for (int i = 0; i < 15; ++i)

list4.pushBack(rand() % 10);

cout << string(10, ' ');

List::print(list4.reverse());

List::print(list3.add(list4).reverse());

list.clear();

list1.clear();

list2.clear();

list3.clear();

list4.clear();

return 0;

}

<https://github.com/artificial-intelligence-department/ai_programming_playground/pull/1127/files#diff-22342bade31ee6a8865c87504af5374d68d8d2b7d7d20f0c46a568204ab7b090>

Завдання №4.2 Class Practice Task 4-5

#include <iostream>

using namespace std;

template <typename T>

class Tree {

private:

struct Node {

T data;

Node\* right;

Node\* left;

};

Node\* root;

bool insertNode(Node\*& current, const T& value) {

if (current == nullptr) {

current = new Node{value};

return true;

} else if (value < current->data) {

return insertNode(current->left, value);

} else if (value > current->data) {

return insertNode(current->right, value);

} else {

return false;

}

}

bool containsNode(const Node\* current, const T& value) const {

if (current == nullptr) {

return false;

} else if (value == current->data) {

return true;

} else if (value < current->data) {

return containsNode(current->left, value);

} else {

return containsNode(current->right, value);

}

}

size\_t sizeNode(const Node\* node) const {

return node == nullptr ? 0 : sizeNode(node->left) + 1 + sizeNode(node->right);

}

void printNode(const Node\* node) const {

if (node != nullptr) {

printNode(node->left);

cout << node->data << " ";

printNode(node->right);

}

}

void printNode(const Node\* node, ostream& os) const {

if (node != nullptr) {

printNode(node->left, os);

os << node->data << " ";

printNode(node->right, os);

}

}

void mirrorNode(Node\* node) {

if (node == nullptr) {

return;

}

Node\* temp = node->left;

node->left = node->right;

node->right = temp;

mirrorNode(node->left);

mirrorNode(node->right);

}

static void copyNode(Node\*& node1, Node\* const& node2){

if (node2 == nullptr){

return;

}

node1 = new Node{node2->data};

copyNode(node1->left, node2->left);

copyNode(node1->right, node2->right);

}

Tree create\_mirrored\_tree(Node\* node){

Tree mirrored;

copyNode(mirrored.root, node);

mirrorNode(mirrored.root);

return mirrored;

}

int tree\_sum(Node\* node){

if (node == nullptr) return 0;

if (node->left == nullptr && node->right == nullptr) return node->data;

return node->data = tree\_sum(node->left) + tree\_sum(node->right);

};

public:

Tree()

: root(nullptr) {}

Tree(const Tree& other)

: root(nullptr) {

copyNode(this->root, other.root);

}

bool insert(const T& value) {

return insertNode(root, value);

}

[[nodiscard]] bool contains(const T& value) const {

return containsNode(root, value);

}

[[nodiscard]] size\_t size() const {

return sizeNode(root);

}

void print() const {

printNode(root);

cout << endl;

}

Tree mirror() {

return create\_mirrored\_tree(root);

}

void sum() {

tree\_sum(root);

}

friend ostream& operator<<(ostream& os, const Tree& tree) {

tree.printNode(tree.root, os);

return os;

}

};

int main() {

Tree<int> tree;

int a[10] = {5,6,3,7,8,9,4,2,5,1};

for (int i : a)

tree.insert(i);

cout << tree << endl;

cout << tree.mirror() << endl;

cout << tree << endl;

tree.sum();

cout << tree;

return 0;

}

<https://github.com/artificial-intelligence-department/ai_programming_playground/pull/1127/files#diff-720e8c268706f810a50e566665f0cf2dc0aa79327b39f3d0b5b2bf476d7588d8>

Завдання № 5 - Self Practice Task

#include <iostream>

using namespace std;

int main() {

int n, k;

cin >> n >> k;

int a[n];

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

cin >> a[i];

int count = 0;

int count\_max = 0;

for (int i : a) {

if (i >= k) {

count++;

if (count >= count\_max) count\_max = count;

}

else count = 0;

}

cout << count\_max;

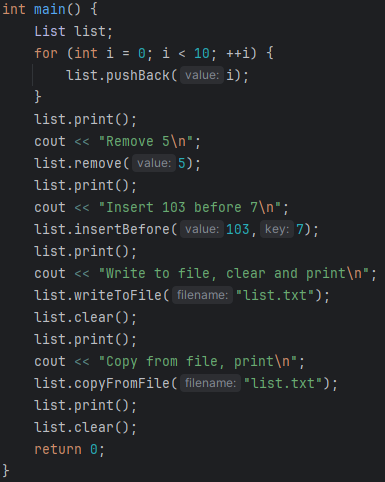
return 0;

}

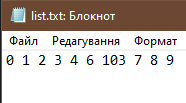
<https://github.com/artificial-intelligence-department/ai_programming_playground/pull/1127/files#diff-b1d4fbcd4abc4928644e050599ac4e0846445828a0aed9719bb31ff7863e2ffd>

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

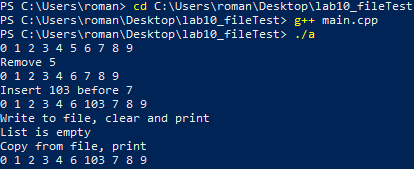
Завдання №1 VNS Lab 10



*Рисунок 2 Виклики функцій завдання №1*



*Рисунок 3 Файл list.txt після виконання завдання №1*



*Рисунок 4 Результат виконання завдання №1*

Час затрачений на виконання завдання: 3год

Завдання №2 Algotester Lab 5



*Рисунок 5 Результат виконання завдання №2*

Час затрачений на виконання завдання: 2год

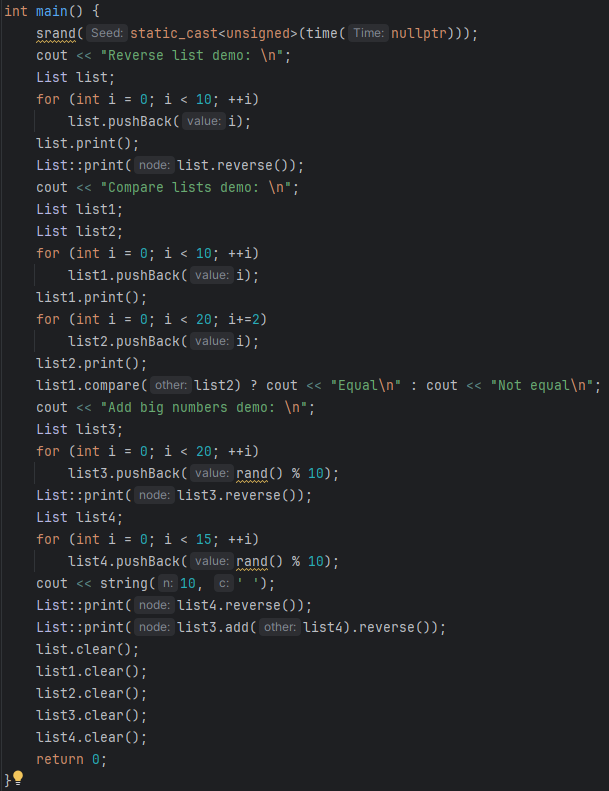
Завдання №3 Algotester Lab 7-8

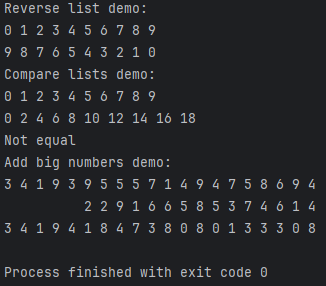


*Рисунок 6 Результат виконання завдання №3*

Час затрачений на виконання завдання: 4год

# Завдання №4.1 Class Practice Task 1-3

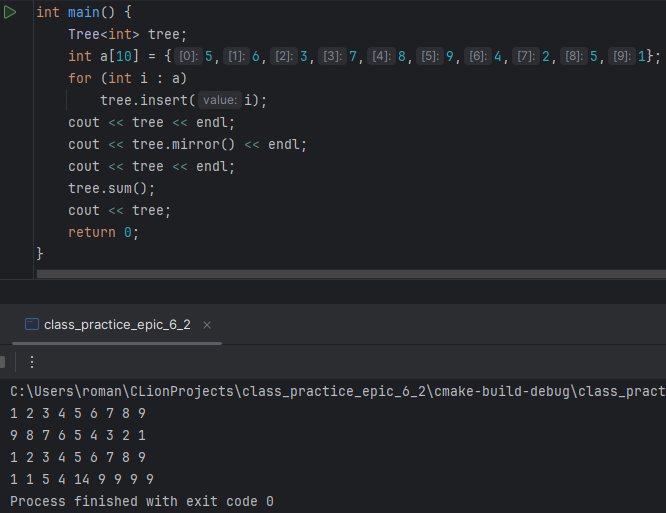
  
*Рисунок 7 Виклики функцій завдання №4.1*



*Рисунок 8 Результат виконання завдання №4.1*

Час затрачений на виконання завдання: 2год

Завдання №4.2 Class Practice Task 4-5



*Рисунок 9 Виклики функцій та результат виконання завдання №4.2*

Час затрачений на виконання завдання: 2год

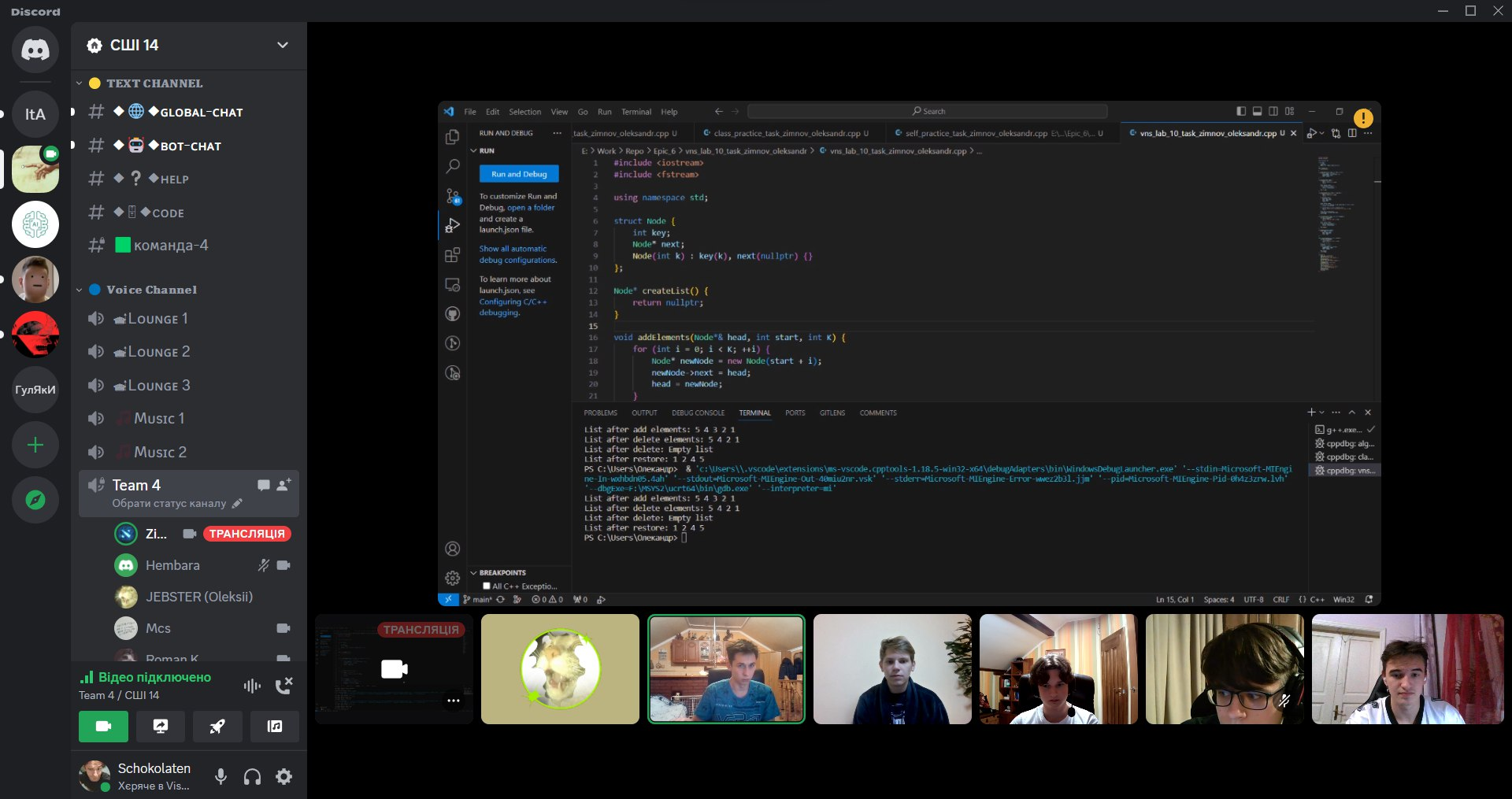
Завдання № 5 - Self Practice Task



*Рисунок 10 Результат виконання завдання №5*

Час затрачений на виконання завдання: 15 хв

## **6. Кооперація з командою:**

* 

# **Висновки:**

Протягом шостого епіку я навчився працювати з динамічними структурами