Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту



**Звіт**

про виконання

**Лабораторних та практичних робіт №6**

***з дисципліни:*** «Мови та парадигми програмування»

***з розділу***: «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур»

до:

ВНС Лабораторної Роботи № 10

Алготестер Лабораторної Роботи № 5

Алготестер Лабораторної Роботи № 7-8

Практичних Робіт № 6

***Виконав:***

студент групи ШІ-11

Грицеляк Маркіян Орестович

# **Тема роботи:**

Динамічні структури даних та алгоритми їх обробки. Однозв’язний та двозв’язний список. Стек, черга та дерево.

# **Мета роботи:**

Ознайомитись з динамічними структурами даних та попрацювати з базовими алгоритмами їх обробки. Реалізувати однозв’язний та двозв’язний списки та створити функції для їх обробки. Ознайомитися з деревами та реалізувати бінарне дерево. Ознайомитися з стеком та чергою

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Стек. Черга.
* Тема №2: Однозв’язний та двозв’язний списки.
* Тема №3: Дерева. Бінарне дерево.
* Тема №4: Класи. Шаблони класів.
* Тема №5: Алгоритми обробки динамічних структур даних.

1. Індивідуальний план опрацювання теорії:

* Тема №1: Стек. Черга.
  + Джерела Інформації
    - Відео.

<https://www.youtube.com/watch?v=ZYvYISxaNL0&t=462s&ab_channel=%D0%91%D0%BB%D0%BE%D0%B3%D0%B0%D0%BD%E2%A6%81%D0%A3%D1%80%D0%BE%D0%BA%D0%B8%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F>

* + - Стаття.

<https://acode.com.ua/urok-111-stek-i-kupa/>

* + Що опрацьовано:
    - Стек і черга як динамічні структури даних, різниця між ними
    - Двобічна та пріоритетна черга
  + Статус: Ознайомлений
  + Початок опрацювання теми: 01.12.2023
  + Звершення опрацювання теми: 22.12.2023
* Тема №2: Однозв’язний та двозв’язний списки.
  + Джерела Інформації:
    - Відео.

<https://www.youtube.com/watch?v=-25REjF_atI&ab_channel=%D0%91%D0%BB%D0%BE%D0%B3%D0%B0%D0%BD%E2%A6%81%D0%A3%D1%80%D0%BE%D0%BA%D0%B8%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F>

<https://www.youtube.com/watch?v=QLzu2-_QFoE&t=1328s&ab_channel=%D0%91%D0%BB%D0%BE%D0%B3%D0%B0%D0%BD%E2%A6%81%D0%A3%D1%80%D0%BE%D0%BA%D0%B8%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F>

* + - Стаття.

<https://www.wikiwand.com/uk/%D0%97%D0%B2'%D1%8F%D0%B7%D0%B0%D0%BD%D0%B8%D0%B9_%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA>

* + Що опрацьовано:
    - Зв’язані списки. Різниця між однозв’язним та двозв’язними списками
  + Статус: Ознайомлений
  + Початок опрацювання теми: 02.12.2023
  + Звершення опрацювання теми: 22.12.2023
* Тема №3: Дерева. Бінарне дерево.
  + Джерела Інформації:
    - Відео.

<https://www.youtube.com/watch?v=o5XvtIPr1Ns&t=437s&ab_channel=%D0%86%D1%80%D0%B8%D0%BD%D0%B0%D0%9E%D0%BB%D1%96%D0%B9%D0%BD%D0%B8%D0%BA>

* + Що опрацьовано:
    - Дерева як структура даних
    - Бінарні дерева
    - Обходи дерева
  + Статус: Ознайомлений
  + Початок опрацювання теми: 03.12.2023
  + Звершення опрацювання теми: 22.12.2023
* Тема №4 Класи. Шаблони класів.
  + Джерела Інформації:
    - Відео.

<https://www.youtube.com/watch?v=a5jigk0Tw50&t=282s&ab_channel=%D0%91%D0%BB%D0%BE%D0%B3%D0%B0%D0%BD%E2%A6%81%D0%A3%D1%80%D0%BE%D0%BA%D0%B8%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F>

* + - Стаття.

<https://acode.com.ua/uroki-po-cpp/>

* + Що опрацьовано:
    - Класи. Шаблони класів
    - Специфікатори доступу (private, public)
    - Конструктори та деструктори
    - Дружні функції
  + Статус: Ознайомлений
  + Початок опрацювання теми: 04.12.2023
  + Звершення опрацювання теми: 22.12.2023
* Тема №4 Алгоритми обробки динамічних структур даних.
  + Джерела Інформації:
    - Відео.
    - Стаття.

<https://www.bestprog.net/uk/2022/02/16/c-linear-doubly-linked-bidirectional-list-general-concepts-ua/>

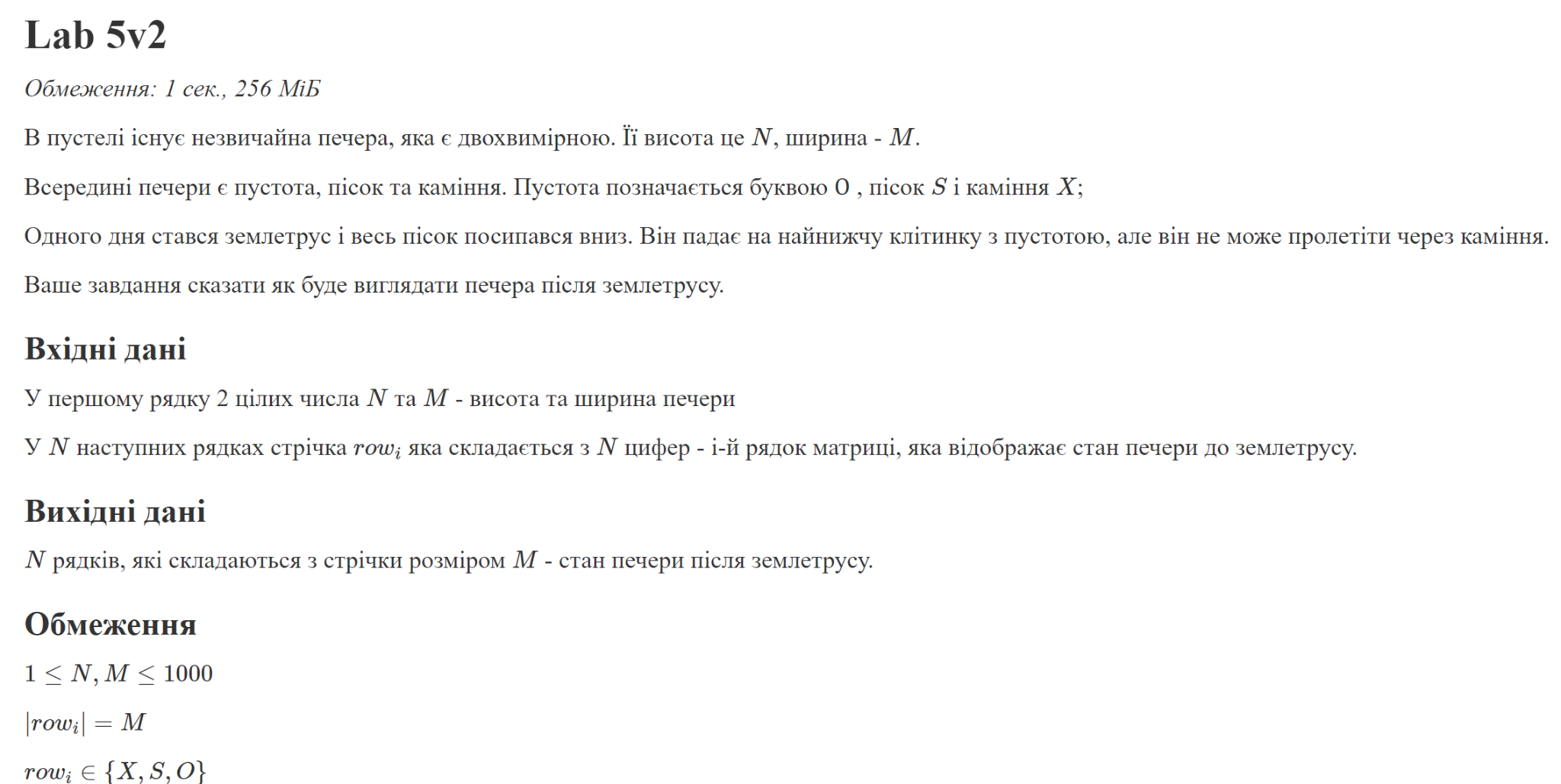
<https://www.bestprog.net/uk/2022/02/13/c-an-example-of-the-implementation-of-a-linear-singly-linked-list-ua/>

* + Що опрацьовано:
    - Алгоритми обробки для динамічних структур даних (вставка елемента у чергу, списки, дерево, стек, видалення елемента…)
    - Бінарний пошук
  + Статус: Ознайомлений
  + Початок опрацювання теми: 03.12.2023
  + Звершення опрацювання теми: 22.12.2023

# **Виконання роботи:**

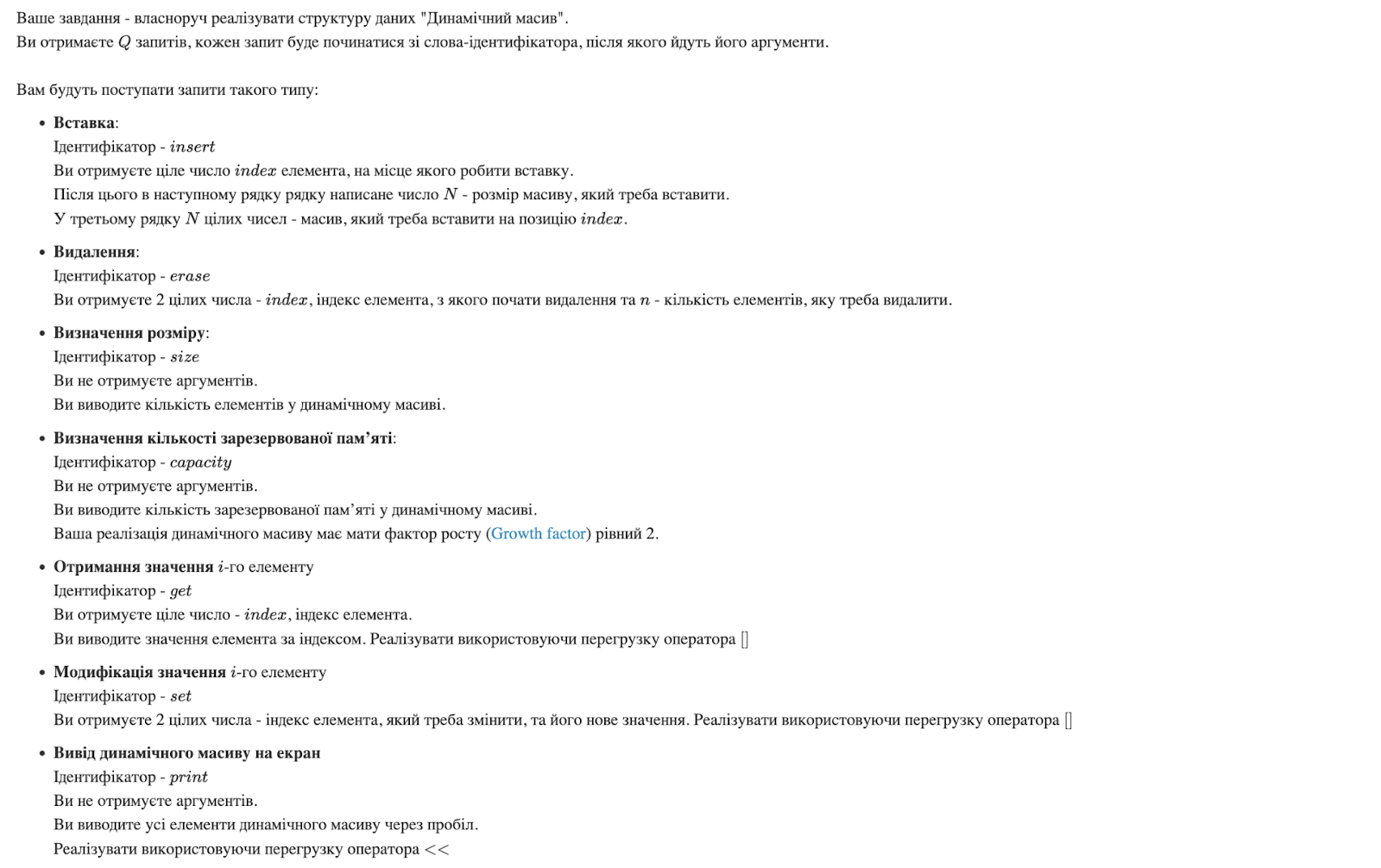
## **1. Опрацювання завдання та вимог до програм та середовища**

* Lab# programming: Algotester Lab 5

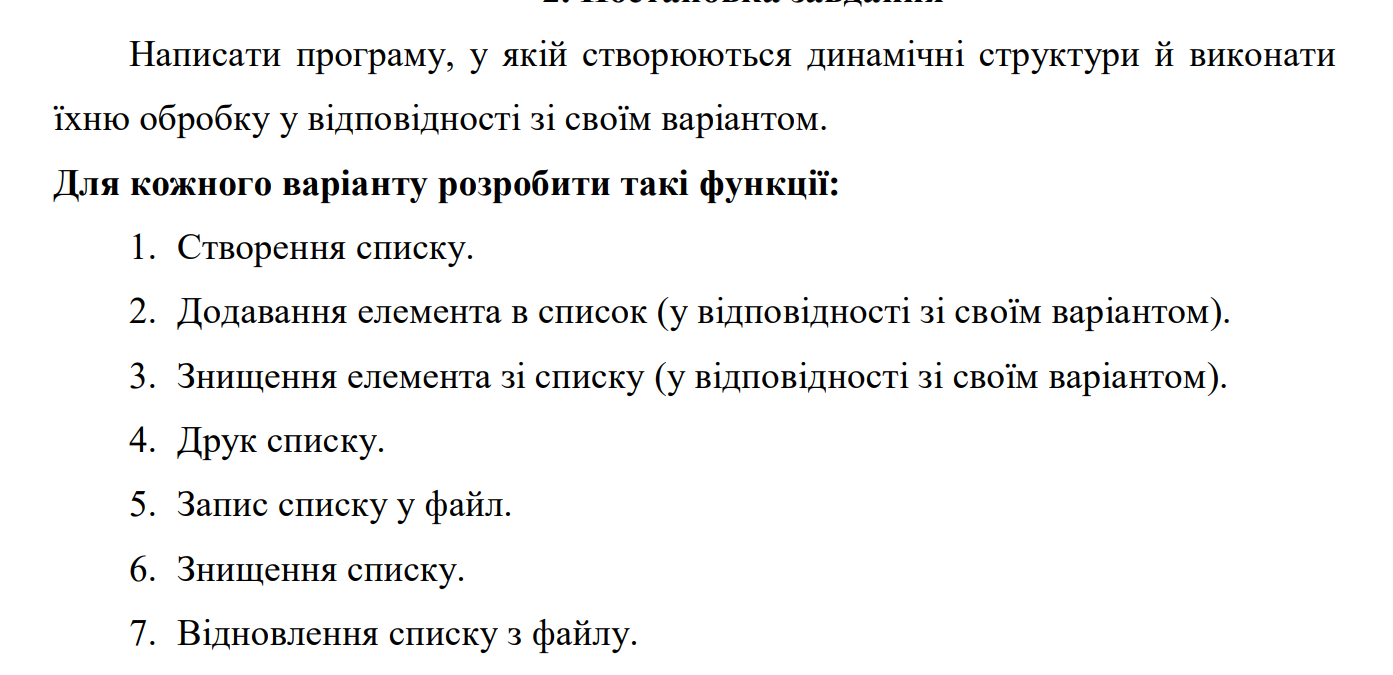


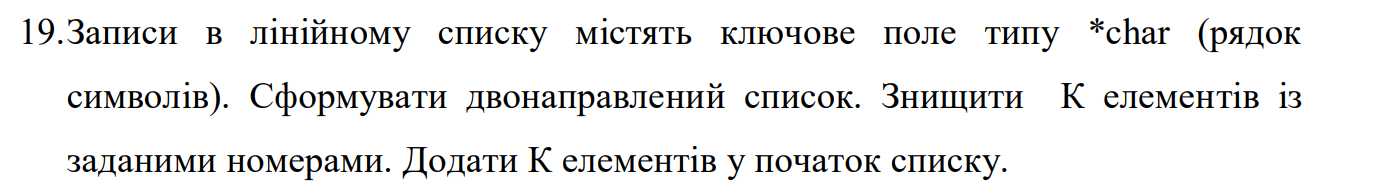
*Рисунок Lab# programming: Algotester Lab 5*

* Lab# programming: Algotester Lab 7-8

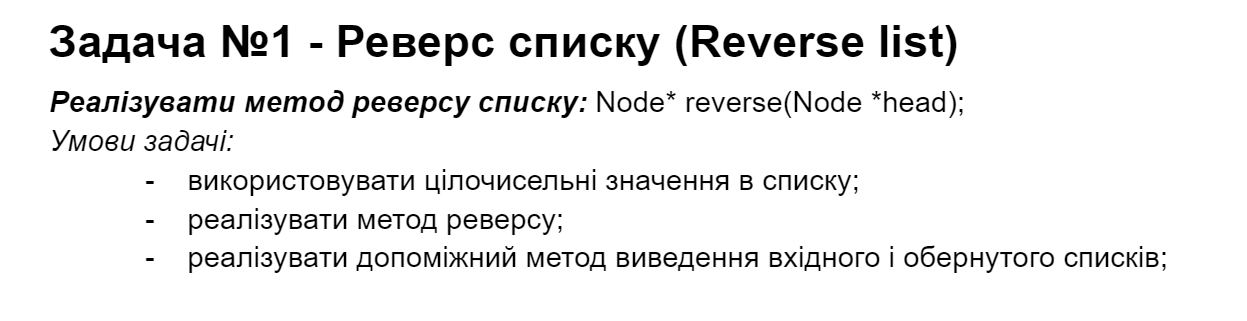


*Lab# programming: Algotester Lab 7-8*

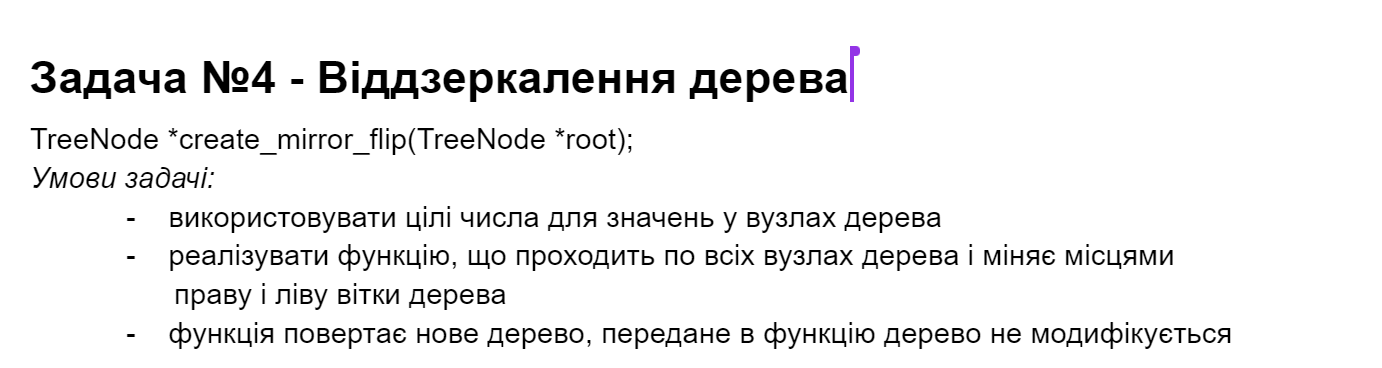




*Lab# programming: VNS Lab 10*



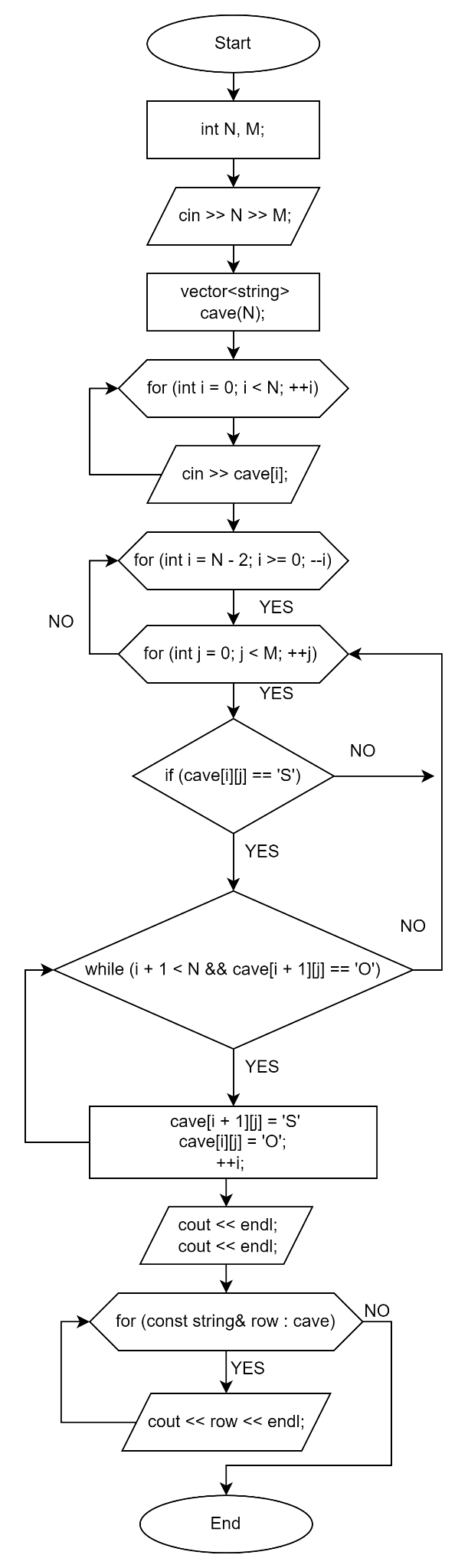
*Class practice task 1*



*Class practice task 4*

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

* Lab# programming: Algotester Lab 5



*Lab# programming: Algotester lab 5*

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

Використано налаштування з епіку 1

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

* Lab# programming: VNS lab 10

#include <iostream>

#include <fstream>

#include <cstring>

#include <vector>

using namespace std;

struct Node

{

char\* key;

Node\* next;

Node\* prev;

Node(const char\* k) : key(strdup(k)), next(nullptr), prev(nullptr) {}

~Node() {

free(key);

}

int getIndex() const {

int index = 0;

const Node\* current = this;

while (current != nullptr) {

if (strcmp(current->key, key) == 0) {

return index;

}

current = current->next;

index++;

}

return -1;

}

};

class DoublyLinkedList

{

private:

Node\* head;

Node\* tail;

public:

DoublyLinkedList() : head(nullptr), tail(nullptr) {}

~DoublyLinkedList() {

destroyList();

}

void createList() {

destroyList();

head = tail = nullptr;

}

void addNode(const char\* key) {

Node\* newNode = new Node(key);

if (!head) {

head = tail = newNode;

} else {

tail->next = newNode;

newNode->prev = tail;

tail = newNode;

}

}

void removeNodes(int\* indices, int count) {

vector<int> indicesToRemove;

Node\* current = head;

int index = 0;

while (current) {

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

if (indices[i] == index) {

indicesToRemove.push\_back(index);

}

}

current = current->next;

++index;

}

}

void removeNodeByIndex(int index) {

Node\* current = head;

int currentIndex = 0;

while (current && currentIndex < index) {

current = current->next;

++currentIndex;

}

if (current && currentIndex == index) {

removeNode(current);

}

}

void addToStart(int k, const char\* key)

{

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

{

Node\* newNode = new Node(key);

if (!head)

{

head = tail = newNode;

}

else

{

newNode->next = head;

head->prev = newNode;

head = newNode;

}

}

}

void printList() const

{

Node\* current = head;

while (current)

{

cout << current->key << " ";

current = current->next;

}

cout << endl;

}

void writeToFile(const string& filename) const

{

ofstream file(filename);

if (!file.is\_open())

{

cerr << "Error opening file " << filename << " for writing!" << endl;

return;

}

Node\* current = head;

while (current)

{

file << current->key << endl;

current = current->next;

}

file.close();

}

void restoreFromFile(const string& filename)

{

ifstream file(filename);

if (!file.is\_open())

{

cerr << "Error opening file " << filename << " for reading!" << endl;

return;

}

destroyList();

char buffer[256];

while (file.getline(buffer, sizeof(buffer)))

{

addNode(buffer);

}

file.close();

}

private:

void destroyList()

{

Node\* current = head;

while (current)

{

Node\* nextNode = current->next;

delete current;

current = nextNode;

}

head = tail = nullptr;

}

void removeNode(Node\* node)

{

if (node->prev)

{

node->prev->next = node->next;

}

else

{

head = node->next;

}

if (node->next)

{

node->next->prev = node->prev;

}

else

{

tail = node->prev;

}

delete node;

}

};

int main() {

DoublyLinkedList myList;

const string path = "C:/Users/Gil/OneDrive/Documents/GitHub/desktop-tutorial/Programming/Code/Epic 6/";

const string filename = "myList.txt";

string file = path+filename;

myList.createList();

myList.addNode("One");

myList.addNode("Two");

myList.addNode("Three");

myList.addNode("Four");

myList.addNode("Five");

cout << "Original list:" << endl;

myList.printList();

int count1;

cout<<"enter amount of elements you want to remove(no more than 5): ";

cin>>count1;

for(int i=0; i<count1; i++)

{

cout<<"Enter element you want to remove: ";

int index;

cin>>index;

myList.removeNodeByIndex(index-1);

}

cout<<"Enter amount of elements 'Zero' you want to add :";

int count2;

cin>>count2;

myList.addToStart(count2, "Zero");

cout << "Modified list:" << endl;

myList.printList();

myList.writeToFile(file);

myList.restoreFromFile(file);

cout << "Restored list:" << endl;

myList.printList();

return 0;

}

* Lab# programming: Algotester Lab 5

#include <iostream>

#include <vector>

using namespace std;

int main() {

int N, M;

cin >> N >> M;

vector<string> cave(N);

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

cin >> cave[i];

}

for (int i = N - 2; i >= 0; --i) {

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

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

while (i + 1 < N && cave[i + 1][j] == 'O') {

cave[i + 1][j] = 'S';

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

++i;

}

}

}

}

cout << endl;

cout<<endl;

for (const string& row : cave) {

cout << row << endl;

}

return 0;

}

* Lab# programming: Algotester Lab 7-8

#include <iostream>

using namespace std;

template <typename T>

class DynamicArray

{

private:

T\* arr;

int size;

int capacity;

public:

DynamicArray()

{

arr = nullptr;

size = 0;

capacity = 1;

}

~DynamicArray()

{

delete[] arr;

}

void insert(int index, int N, const T\* values)

{

if (index < 0 || index > size) {

cerr << "Invalid index for insert!" << endl;

return;

}

if (size + N >= capacity)

{

while(size + N >= capacity)

{

capacity \*= 2;

resize(capacity);

}

}

for (int i = size - 1; i >= index; --i)

{

arr[i + N] = arr[i];

}

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

arr[index + i] = values[i];

}

size += N;

}

void erase(int index, int n)

{

if (index < 0 || index >= size || n < 0)

{

cerr << "Invalid index or count for erase!" << endl;

return;

}

for (int i = index + n; i < size; ++i)

{

arr[i - n] = arr[i];

}

size -= n;

}

int getSize() const

{

return size;

}

int getCapacity() const

{

return capacity;

}

T& operator[](int index)

{

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

{

cerr << "Index out of bounds!" << endl;

exit(1);

}

return arr[index];

}

void set(int index, T value)

{

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

{

cerr << "Index out of bounds!" << endl;

return;

}

arr[index] = value;

}

void print() const

{

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

{

cout << arr[i] << ' ';

}

cout << endl;

}

private:

void resize(int newCapacity)

{

T\* newArr = new T[newCapacity];

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

{

newArr[i] = arr[i];

}

delete[] arr;

arr = newArr;

capacity = newCapacity;

}

};

int main()

{

int Q;

cin >> Q;

DynamicArray<int> dynamicArray;

while (Q--)

{

string operation;

cin >> operation;

if (operation == "insert")

{

int index, N;

cin >> index >> N;

int\* values = new int[N];

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

{

cin >> values[i];

}

dynamicArray.insert(index, N, values);

delete[] values;

}

else if (operation == "erase")

{

int index, n;

cin >> index >> n;

dynamicArray.erase(index, n);

}

else if (operation == "size")

{

cout << dynamicArray.getSize() << endl;

}

else if (operation == "capacity")

{

cout << dynamicArray.getCapacity() << endl;

}

else if (operation == "get")

{

int index;

cin >> index;

cout << dynamicArray[index] << endl;

}

else if (operation == "set")

{

int index, value;

cin >> index >> value;

dynamicArray.set(index, value);

}

else if (operation == "print")

{

dynamicArray.print();

}

}

return 0;

}

* Practice# programming: Class Practice Task1

#include <iostream>

using namespace std;

struct Node {

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;

}

void printList(Node\* head) {

while (head != nullptr) {

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

head = head->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);

cout << "Original list: ";

printList(head);

head = reverse(head);

cout << "Reversed list: ";

printList(head);

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

return 0;

}

* Practice# programming: Class Practice Task 4

#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>

TreeNode<T>\* insert(TreeNode<T>\* root, T value) {

if (root == nullptr) {

return new TreeNode<T>(value);

}

if (value < root->data) {

root->left = insert(root->left, value);

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

root->right = insert(root->right, value);

}

return root;

}

template <typename T>

void printInOrder(TreeNode<T>\* root) {

if (root != nullptr) {

printInOrder(root->left);

cout << root->data << " ";

printInOrder(root->right);

}

}

template <typename T>

TreeNode<T>\* mirrorFlip(TreeNode<T>\* root) {

if (root == nullptr) {

return nullptr;

}

TreeNode<T>\* temp = root->left;

root->left = mirrorFlip(root->right);

root->right = mirrorFlip(temp);

return root;

}

int main() {

TreeNode<int>\* root = nullptr;

root = insert(root, 10);

insert(root, 5);

insert(root, 15);

insert(root, 3);

insert(root, 7);

insert(root, 6);

insert(root, 9);

cout << "Original Binary Tree (in-order): ";

printInOrder(root);

cout <<endl;

TreeNode<int>\* mirroredRoot = mirrorFlip(root);

cout << "Mirrored Binary Tree (in-order): ";

printInOrder(mirroredRoot);

cout << endl;

return 0;

// 7

// / \

// 5 10 - Перше дерево

// / \ / \

// 3 6 9 15

//

// 7

// / \

// 10 5 - Відзеркалене дерево

// / \ / \

// 15 9 6 3

}

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

* Lab# programming: VNS lab 10

Original list:

One Two Three Four Five

enter amount of elements you want to remove(no more than 5): 2

Enter element you want to remove: 1

Enter element you want to remove: 3

Enter amount of elements 'Zero' you want to add :3

Modified list:

Zero Zero Zero Two Three Five

Restored list:

Zero Zero Zero Two Three Five

Час витрачений на виконання завдання: 3 години

* Lab# programming: Algotester Lab 5

5 5

SSOOO

OXSOO

OOXSO

OOOXS

XOOOO

OSOOO

OXSOO

OOXSO

SOOXO

XOOOS

Час витрачений на виконання завдання: 1 година



*Algotester lab 5*

* Lab# programming: Algotester Lab 7-8

12

size

0

capacity

1

insert 0 2

100 100

size

2

capacity

4

insert 0 2

102 102

size

4

capacity

8

insert 0 2

103 103

size

6

capacity

8

print

103 103 102 102 100 100

Час витрачений на виконання завдання: 4 години



*Algotester lab 7-8*

* Practice# programming: Class Practice Task 1

Original list: 1 2 3 4 5

Reversed list: 5 4 3 2 1

1 -> 2 -> 3 -> 4 -> 5

5 -> 4 -> 3 -> 2 -> 1

Час витрачений на виконання завдання: 2 години

* Practice# programming: Class Practice Task 4

Original Binary Tree (in-order): 3 5 6 7 9 10 15

Mirrored Binary Tree (in-order): 15 10 9 7 6 5 3

// 7

// / \

// 5 10 - Перше дерево

// / \ / \

// 3 6 9 15

//

//

// 7

// / \

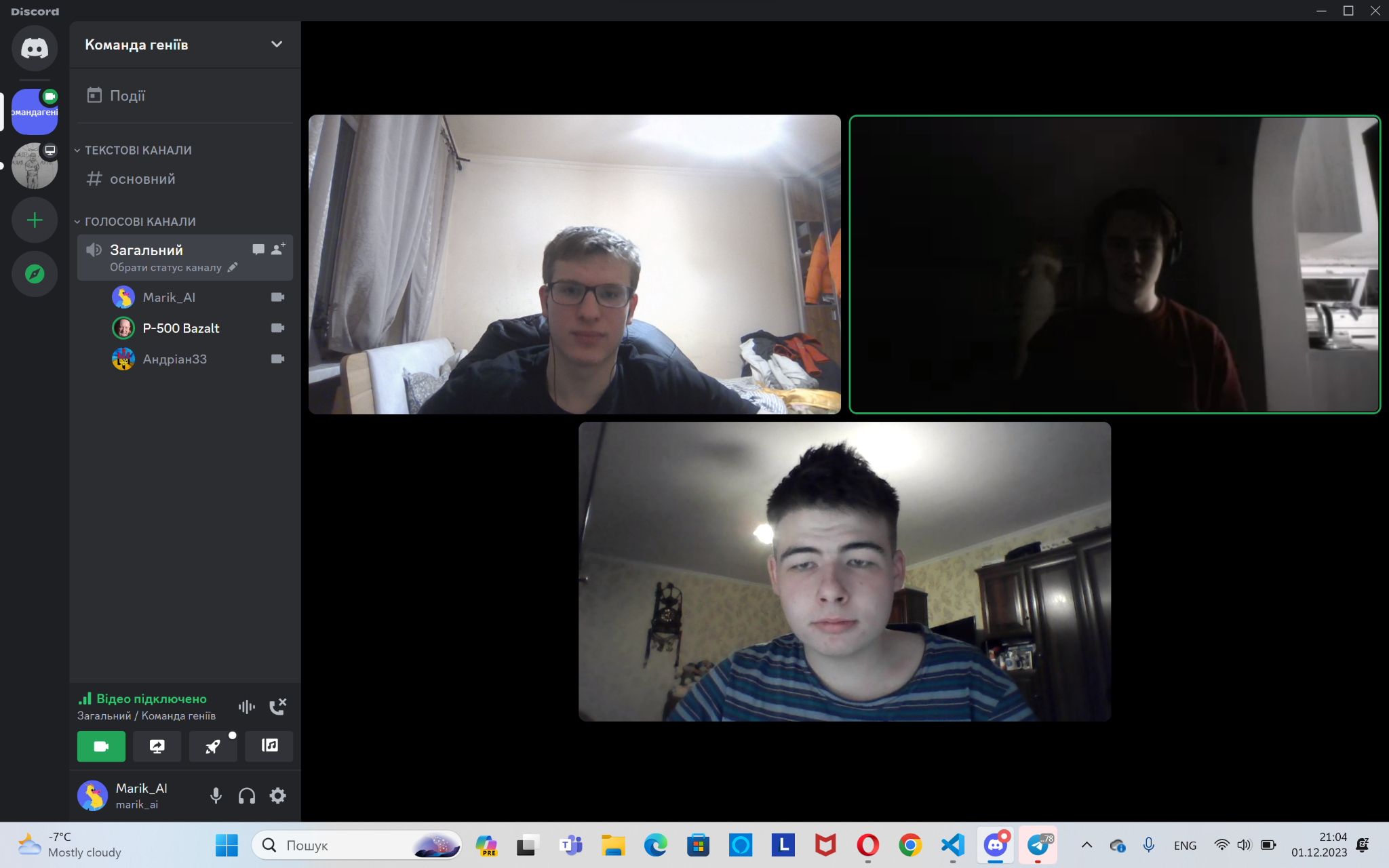
// 10 5 - Відзеркалене дерево

// / \ / \

// 15 9 6 3

Час витрачений на виконання завдання: 2 години

## **6. Кооперація з командою:**



*chilling with teammates*

# **Висновки:**

Опрацьовано динамічні структури даних та реалізовано деякі з них. Ознайомився з деревами, чергою, двозв’язним і однозв’язним списками та іншими структурами даних. Використано динамічні структури даних у виконанні завдань з лабораторної роботи.Нарешті це закінчилось!!!