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

Сокаль Олег Ігорович

# **Тема роботи:**

# Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур

# **Мета роботи:**

# **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
    - Лекції
    - Лекційний матеріал на ВНС
  + Що опрацьовано:
    - Черга
    - Стек
    - Списки
    - Дерево
  + Статус: Ознайомлений
  + Початок опрацювання теми: 30.11.2023
  + Звершення опрацювання теми: 01.12.2023
* Тема №2: Алгоритми обробки динамічних структур
  + Джерела Інформації:
    - Лекції
    - Лекційний матеріал на ВНС
  + Що опрацьовано:
    - Алгоритми для вставки, вилучення, пошуку та інших операцій
    - Linked list
  + Статус: Ознайомлений
  + Початок опрацювання теми: 01.12.2023
  + Звершення опрацювання теми: 02.12.2023

# **Виконання роботи:**

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

Завдання №1 - Theory Education Activities

* Вивчення теорії

Завдання №2 - Requirements management (understand tasks) and design activities (draw flow diagrams and estimate tasks 3-7)

* Створення діаграм для програм в draw io

Завдання №3 - Lab# programming: VNS Lab 10

* Варіант № 4
* Деталі завдання: Написати програму, у якій створюються динамічні структури й виконати їхню обробку у відповідності зі своїм варіантом. Для кожного варіанту розробити такі функції: створення списку, додавання елемента в список (у відповідності зі своїм варіантом), знищення елемента зі списку (у відповідності зі своїм варіантом), друк списку, запис списку у файл, знищення списку, відновлення списку з файлу. Варіант: Записи в лінійному списку містять ключове поле типу int. Сформувати однонаправлений список. Знищити з нього елемент із заданим номером, додати К елементів, починаючи із заданого номера
* Важливі деталі для врахування в імплементації програми: Функція може створювати порожній список, а потім додавати в нього елементи, після виведення функцій потрібно роздрукувати список, якщо при виведення список порожній, то потрібно вивести відповідне повідомлення

Завдання №4 - Lab# programming: Algotester Lab 5

* Варіант № 3
* Деталі завдання: 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. Клітинкі 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.
* Важливі деталі для врахування в імплементації програми: 1≤N,M≤103, 1≤x≤N, 1≤y≤M, спочатку варто знайти число найвищої точки, після цього згенерувати гору буде простіше

Завдання №5 - Lab# programming: Algotester Lab 7-8

* Варіант № 2
* Деталі завдання: Ваше завдання - власноруч реалізувати структуру даних "Динамічний масив".  
  Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його аргументи. Вам будуть поступати запити такого типу: **Вставка**: Ідентифікатор – insert. Ви отримуєте ціле число index елемента, на місце якого робити вставку. Після цього в наступному рядку рядку написане число N - розмір масиву, який треба вставити. У третьому рядку N цілих чисел - масив, який треба вставити на позицію index. **Видалення**: Ідентифікатор – erase. Ви отримуєте 2 цілих числа - index, індекс елемента, з якого почати видалення та n - кількість елементів, яку треба видалити. **Визначення розміру**: Ідентифікатор – size. Ви не отримуєте аргументів. Ви виводите кількість елементів у динамічному масиві. **Визначення кількості зарезервованої пам’яті**: Ідентифікатор – capacity. Ви не отримуєте аргументів. Ви виводите кількість зарезервованої пам’яті у динамічному масиві. Ваша реалізація динамічного масиву має мати фактор росту ([Growth factor](https://en.wikipedia.org/wiki/Dynamic_array#Growth_factor)) рівний 2. **Отримання значення** i-го елементу: Ідентифікатор – get. Ви отримуєте ціле число - index, індекс елемента. Ви виводите значення елемента за індексом. Реалізувати використовуючи перегрузку оператора [][]. **Модифікація значення** i-го елементу: Ідентифікатор – set. Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення. Реалізувати використовуючи перегрузку оператора [][]. **Вивід динамічного масиву на екран**: Ідентифікатор – print. Ви не отримуєте аргументів. Ви виводите усі елементи динамічного масиву через пробіл. Реалізувати використовуючи перегрузку оператора <<
* Важливі деталі для врахування в імплементації програми: 0≤Q≤105, 0≤li≤105, ∥l∥≤105. Для отримання 100% балів ця структура має бути написана як [шаблон класу](https://cplusplus.com/doc/oldtutorial/templates/), у якості параметру використати int

Завдання №6 - Practice# programming: Class Practice Task

* Деталі завдання №1: 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
* Деталі завдання №2: bool compare(Node \*h1, Node \*h2); 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
* Важливі деталі для врахування в імплементації програми: Потрібно зробити структуру Node для даних файлу, для кожної дії потрібно створити відповідну функцію

Завдання №7 - Practice# programming: Self Practice Task

* Деталі завдання: Створити структуру, для запам’ятовування інформації про персонажа, з них Ім’я, Швидкість, Втрачене здоров’я, Втрачені рівні заклинання, Втрачені побічні здібності персонажа, Кількість можливостей лікування. Вивести меню, в якому можна створити, видалити, або змінити параметра персонажу. Також програма має вміти вивести всіх персонажів впорядкованих за швидкістю. В кінці програму можна записати в файл. А при запуску програма має зчитувати ці данні
* Важливі деталі для врахування в імплементації програми: Рівнів заклинань є 5 і їх може бути різна кількість, користувач вводить всі “втрачені” параметри як додатні, достатньо просто помножити їх на -1. Індекс персонажа може змінюватись залежно від теперішньої позиції

Завдання №8 - Result Documentation Report and Outcomes Placement Activities (Docs and Programs on GitHub)

* Створення звіту і завантажити його з програмами на GitHub

Завдання №9 - Results Evaluation and Release

* Захист звіту.

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

Програма №1 VNS Lab 10

* Час на виконання: 3 години
* Важливі деталі для врахування в імплементації програми: Завдання можна виконати в одному файлі для того щоб використовувати спільні функції. Записи в списку містять поля типу int, що треба вказати в структурі

Програма №2 Algotester Lab 5

* Час на виконання: 1 година і 30 хвилин
* Важливі деталі для врахування в імплементації програми: Завдяки функції abs можна знайти максимальне значення гори, а потім завдяки цьому числу і масиву можна сформувати всі решта числа

Програма №3 Algotester Lab 7-8

*Рисунок 1: Блок схема до програми №5*

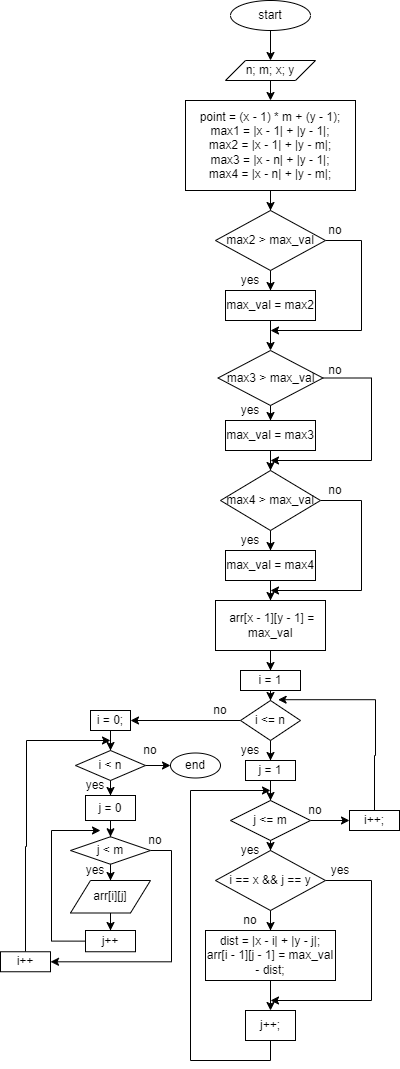
* Час на виконання: 6 годин
* Важливі деталі для врахування в імплементації програми: capacity завжди має бути більше ніж size, якщо ж ні, то потрібно його домножити на growthFactor

Програма №4 Class Practice Task

* Час на виконання: 2 години
* Важливі деталі для врахування в імплементації програми: Для зручності list був створений самою програмою, провірку можна робити до nullptr, що означає кінець списку.

Програма №5 Self Practice Task

* Час на виконання: 3 години
* Важливі деталі для врахування в імплементації програми: Важливо зазначити, що для кожного рівня заклинання є відмінна кількість самих заклинань. Як максимум можна взяти 10 гравців, цього цілком достатньо для списку.



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

Завдання №1: VNS Lab 10

#include <iostream>

using namespace std;

class DblLinkedList {

private:

struct Node {

int data;

Node\* prev;

Node\* next;

};

Node\* head;

Node\* tail;

size\_t size;

public:

// Constructor

DblLinkedList() : head(nullptr), tail(nullptr), size(0) {}

// Destructor to clean up memory

~DblLinkedList() {

Node\* current = head;

while (current != nullptr) {

Node\* nextNode = current->next;

delete current;

current = nextNode;

}

}

// Display the elements in the list

void Show() const {

if (size == 0) {

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

return;

}

Node\* current = head;

while (current != nullptr) {

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

current = current->next;

}

cout << endl;

}

// Add a node with the given value to the back of the list

void PushBack(const int& value) {

if (size == 0) {

head = tail = new Node{value, nullptr, nullptr};

} else {

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

tail = tail->next;

}

size++;

}

// Add a node with the given value to the front of the list

void PushFront(const int& value) {

if (size == 0) {

head = tail = new Node{value, nullptr, nullptr};

} else {

Node\* newHead = new Node{value, nullptr, head};

head->prev = newHead;

head = newHead;

}

size++;

}

// Remove nodes with the given value from the list

void Remove(int value) {

Node\* current = head;

while (current != nullptr) {

if (current->data == value) {

if (head == current) {

if (size == 1) {

delete head;

head = tail = nullptr;

} else {

Node\* tmp = head;

head = head->next;

head->prev = nullptr;

delete tmp;

}

} else if (tail == current) {

Node\* tmp = tail;

tail = tail->prev;

tail->next = nullptr;

delete tmp;

} else {

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

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

delete current;

}

size--;

return;

}

current = current->next;

}

}

};

int main() {

DblLinkedList list;

cout << "List: ";

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

list.PushBack(value);

}

list.Show();

cout << "List after deletion: ";

for (int i = 0; i < 3; i++) {

list.Remove(i);

}

list.Show();

cout << "List after adding elements: ";

for (int i = 0; i < 3; i++) {

list.PushFront(i);

}

list.Show();

return 0;

}

*Рисунок 2: Код завдання з VNS №10*

#include <iostream>

#include <string>

using namespace std;

void swap(char &a, char &b) {

char temp = a;

a = b;

b = temp;

}

int main() {

int n, m;

cin >> n >> m;

char arr[1000][1000];

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

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

cin >> arr[i][j];

}

}

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

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

if (arr[i][j] == 'S' && i + 1 < n && arr[i + 1][j] == 'O') {

swap(arr[i][j], arr[i + 1][j]);

while (i + 2 < n && arr[i + 2][j] == 'O') {

swap(arr[i + 1][j], arr[i + 2][j]);

swap(arr[i][j], arr[i + 1][j]);

i++;

}

}

}

}

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

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

cout << arr[i][j];

}

cout << endl;

}

return 0;

}

*Рисунок 3: Код завдання з Algotester №5*

Завдання №3: Algotester Lab 7-8

#include <iostream>

#include <sstream>

#include <algorithm>

using namespace std;

template <typename T>

class DynamicArray {

private:

T\* arr;

int size;

int capacity;

const int growthFactor = 2;

void resize() {

do {

capacity \*= growthFactor;

} while (capacity <= size);

T\* newArr = new T[capacity];

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

newArr[i] = arr[i];

}

delete[] arr;

arr = newArr;

}

public:

DynamicArray() : size(0), capacity(1) {

arr = new T[capacity];

}

~DynamicArray() {

delete[] arr;

}

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

if (index > size) {

return;

}

size += N;

while (size >= capacity) {

resize();

}

copy\_backward(arr + index, arr + size - N, arr + size);

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

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

}

}

void erase(int index, int n) {

if (index >= size) {

return;

}

int elementsToRemove = min(n, size - index);

for (int i = index; i < size - elementsToRemove; ++i) {

arr[i] = arr[i + elementsToRemove];

}

size -= elementsToRemove;

}

int getSize() const {

return size;

}

int getCapacity() const {

return capacity;

}

T& operator[](int index) {

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

throw out\_of\_range("Index out of range");

}

return arr[index];

}

void set(int index, T value) {

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

throw out\_of\_range("Index out of range");

}

arr[index] = value;

}

void print() const {

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

cout << arr[i] << " ";

}

cout << endl;

}

};

int main() {

int Q;

cin >> Q;

DynamicArray<int> dynamicArray;

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

string operat;

cin >> operat;

if (operat == "insert") {

int index, N;

cin >> index >> N;

int\* values = new int[N];

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

cin >> values[j];

}

dynamicArray.insert(index, N, values);

delete[] values;

}

else if (operat == "get") {

int index;

cin >> index;

try {

cout << dynamicArray[index] << endl;

} catch (const out\_of\_range& e) {

cerr << "Error: " << e.what() << endl;

}

}

else if (operat == "size") {

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

}

else if (operat == "set") {

int index, value;

cin >> index >> value;

try {

dynamicArray.set(index, value);

} catch (const out\_of\_range& e) {

cerr << "Error: " << e.what() << endl;

}

}

else if (operat == "erase") {

int index, n;

cin >> index >> n;

dynamicArray.erase(index, n);

}

else if (operat == "print") {

dynamicArray.print();

}

else if (operat == "capacity") {

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

}

}

return 0;

}

Рисунок 4: Код завдання з Algotester №7-8

Завдання №4: Class Practice Task

#include <iostream>

using namespace std;

class Node {

public:

int value;

Node\* next;

// Constructor to initialize values

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

};

Node\* reverse(Node\* head) {

Node\* current = head;

Node\* prev = nullptr;

Node\* next = nullptr;

while (current != nullptr) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

head = prev;

return head;

}

void printList(Node\* n) {

if (n == nullptr) {

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

return;

}

while (n != nullptr) {

cout << n->value << " ";

n = n->next;

}

}

int main() {

// Initialize nodes with values using constructor

Node\* head = new Node(1);

Node\* second = new Node(2);

Node\* third = new Node(3);

// Link nodes together

head->next = second;

second->next = third;

third->next = nullptr;

cout << "Original list: ";

printList(head);

cout << endl;

cout << "Reversed list: ";

head = reverse(head);

printList(head);

cout << endl;

// Clean up memory

delete head;

delete second;

delete third;

return 0;

}

#include <iostream>

using namespace std;

class Node {

public:

int value;

Node\* next;

// Constructors

Node() : value(0), next(nullptr) {}

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

// Destructor for proper memory cleanup

~Node() {

delete next;

}

};

bool compare(const Node\* a, const Node\* b) {

while (a != nullptr && b != nullptr) {

if (a->value != b->value) {

return false;

}

a = a->next;

b = b->next;

}

return (a == nullptr && b == nullptr);

}

int main() {

// Initialize linked lists using constructors

Node\* head1 = new Node(2);

head1->next = new Node(4);

head1->next->next = new Node(6);

Node\* head2 = new Node(2);

head2->next = new Node(4);

head2->next->next = new Node(7);

// Compare linked lists

if (compare(head1, head2)) {

cout << "Lists are equal" << endl;

} else {

cout << "Lists are not equal" << endl;

}

// Clean up memory

delete head1;

delete head2;

return 0;

}

#include <iostream>

#include <queue>

using namespace std;

struct TreeNode {

int data;

TreeNode\* left;

TreeNode\* right;

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

};

TreeNode\* insert(TreeNode\* root, int value) {

if (root == nullptr) {

return new TreeNode(value);

}

if (value < root->data) {

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

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

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

}

return root; // Return root outside the if-else block

}

void inorderTraversal(TreeNode\* root) {

if (root != nullptr) {

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

}

void tree\_sum(TreeNode\* root) {

if (root == nullptr) {

return;

}

queue<TreeNode\*> nodeQueue;

nodeQueue.push(root);

while (!nodeQueue.empty()) {

TreeNode\* current = nodeQueue.front();

nodeQueue.pop();

if (current->left != nullptr || current->right != nullptr) {

int leftSum = (current->left != nullptr) ? current->left->data : 0;

int rightSum = (current->right != nullptr) ? current->right->data : 0;

current->data = leftSum + rightSum;

}

if (current->left != nullptr) {

nodeQueue.push(current->left);

}

if (current->right != nullptr) {

nodeQueue.push(current->right);

}

}

}

int main() {

TreeNode\* root = nullptr;

int elements[] = {10, 5, 15, 3, 7, 12, 20};

for (int element : elements) {

root = insert(root, element);

}

cout << "Original Tree: ";

inorderTraversal(root);

cout << endl;

tree\_sum(root);

cout << "Tree after sum calculation: ";

inorderTraversal(root);

cout << endl;

return 0;

}

*Рисунок 5: Код програми з практичної роботи*

Завдання №5: Self Practice Task

#include <iostream>

#include <cmath>

#include <vector>

using namespace std;

int main() {

int n, m, max1, max2, max3, max4;

int point, OldX, x, y;

cin >> n >> m;

cin >> x >> y;

vector<vector<int>> arr(n, vector<int>(m, 0));

point = (x - 1) \* m + (y - 1);

max1 = abs(x - 1) + abs(y - 1);

max2 = abs(x - 1) + abs(y - m);

max3 = abs(x - n) + abs(y - 1);

max4 = abs(x - n) + abs(y - m);

int max\_val = max1;

if (max2 > max\_val) {

max\_val = max2;

}

if (max3 > max\_val) {

max\_val = max3;

}

if (max4 > max\_val) {

max\_val = max4;

}

arr[x - 1][y - 1] = max\_val;

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= m; j++) {

if (i == x && j == y) continue;

int dist = abs(x - i) + abs(y - j);

arr[i - 1][j - 1] = max\_val - dist;

}

}

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

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

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

}

cout << endl;

}

return 0;

}

*Рисунок 6: Код програми для самостійного опрацювання*

Посилання на програми в GitHub: <https://github.com/Zimnov1/Repo/tree/main>

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

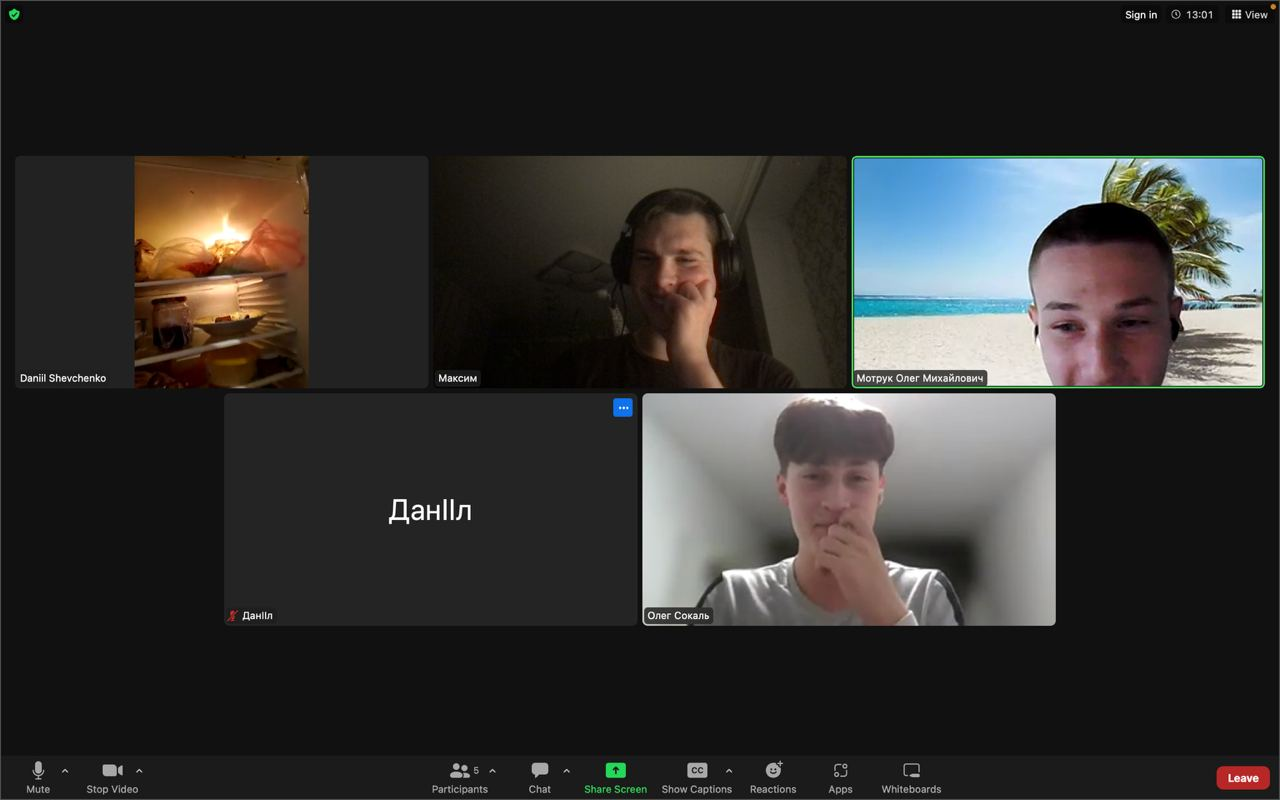
Деталі по виконанню:

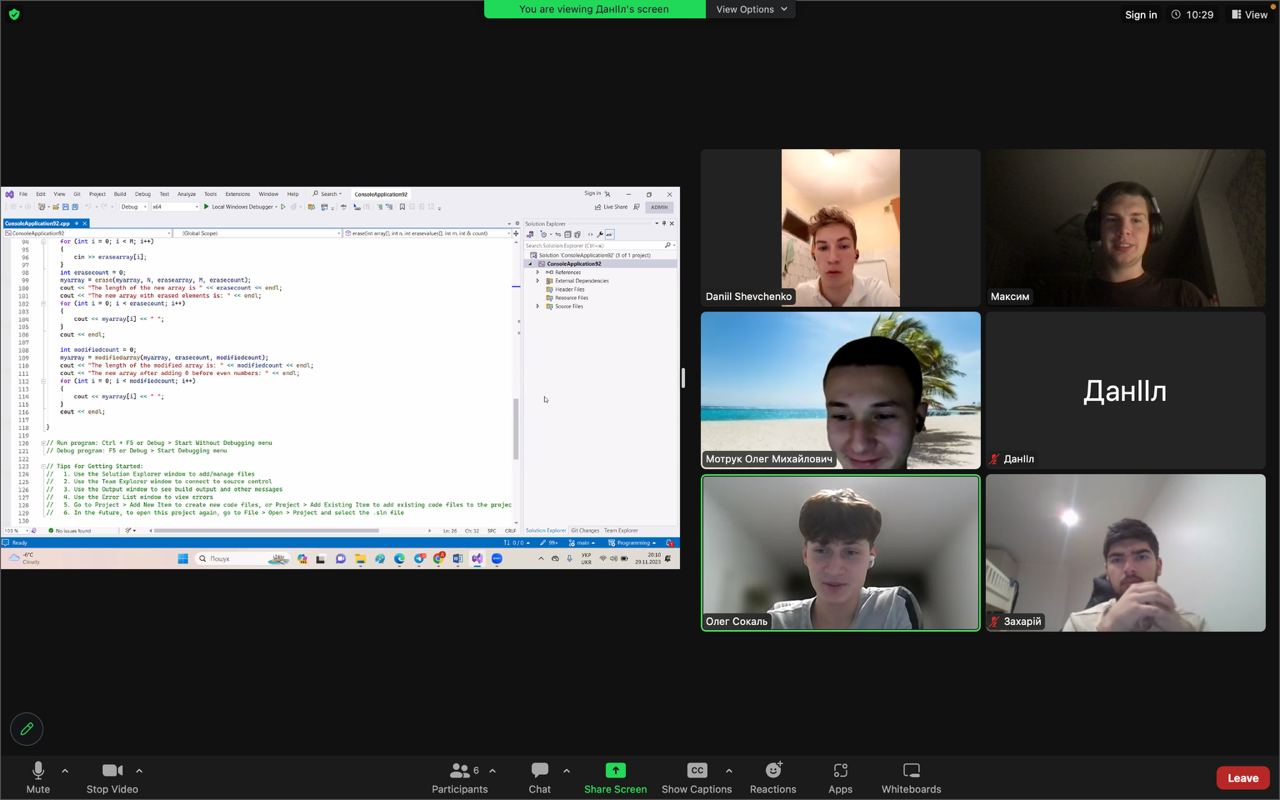
* Потрібно було створити шаблон за замовчуванням
* Програма зберігає дані при виході, щоб запобігти втраті даних

Час затрачений на виконання завдання: 3 години

## **5. Кооперація з командою:**

Знімки конференції:





*Рисунок 12: Знімки зустрічі з командою*

# **Висновки:**

# Протягом лабораторних та практичних робіт №6було вивчено динамічні структури, такі як черга, стек, списки і дерево, а також алгоритми обробки динамічних структур

Всі завдання були відслідковані в дошці на Trello. Були онлайн зустрічі з командою, в яких ми владнали всі непорозуміння.

Посилання на Pull request: https://github.com/artificial-intelligence department/ai\_programming\_playground/pull/687