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:*** «Epic 6. Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур»

***Виконав:***

студент групи ШІ-14

Михальчишин Лук’ян Павлович

**Тема роботи:**

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

**Мета роботи:**

Опанувати навики по таким поняттям як стек, однозв’язні і двозв’язні списки, що таке дерева і для чого вони взагалі придумані, також зрозуміти що таке класи

**Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:
2. Індивідуальний план опрацювання теорії:

* **Тема №1: Cтек**

Джерела Інформації:

* + Курси: <https://youtu.be/NyOjKd5Qruk?si=P1dafCwKQCyfcAOo>
  + Статті: <https://acode.com.ua/urok-111-stek-i-kupa/>
  + Славнозвісний GPT, також консультація з одногрупниками (безкоштовно звичайно)

Статус: Ознайомлений

* **Тема №2: Класи**

Джерела Інформації:

* + Курси: <https://youtu.be/ATNE6AKZ-WU?si=D5kwywQr0JTU0KIs>

<https://youtu.be/ipzULicIS54?si=07sN945GGrGjxNSZ>

* + Статті: <https://acode.com.ua/urok-121-klasy-ob-yekty-i-metody/>
  + Славнозвісний GPT, також консультація з одногрупниками (безкоштовно звичайно)

Статус: Ознайомлений

* **Тема №3: Однозв’язний список**

Джерела Інформації:

* + Курси: <https://youtu.be/-25REjF_atI?si=Do1k54eoRGYbdJbH>
  + Статті: <https://www.bestprog.net/uk/2022/02/11/c-linear-singly-linked-list-general-information-ua/>
  + Славнозвісний GPT, також консультація з одногрупниками (безкоштовно звичайно)

Статус: Ознайомлений

* **Тема №4: Двозв’язний спискок**

Джерела Інформації:

* + Курси: <https://youtu.be/QLzu2-_QFoE?si=NPZVVh-E2F28Ha41>
  + Статті: <https://www.bestprog.net/uk/2022/02/16/c-a-linear-doubly-linked-list-an-example-of-a-template-class-ua/>
  + Славнозвісний GPT, також консультація з одногрупниками (безкоштовно звичайно)

Статус: Ознайомлений

* **Тема №5: Бінарні дерева**

Джерела Інформації:

* + Курси: <https://youtu.be/qBFzNW0ALxQ?si=OD6KKSJ9c078pj9L>
  + Статті: <https://purecodecpp.com/uk/archives/2483>
  + Славнозвісний GPT, також консультація з одногрупниками (безкоштовно звичайно)
  + PS: Не знав що в програмуванні існують дерева, досить смішно звучали слова коріння листки і саме дерево

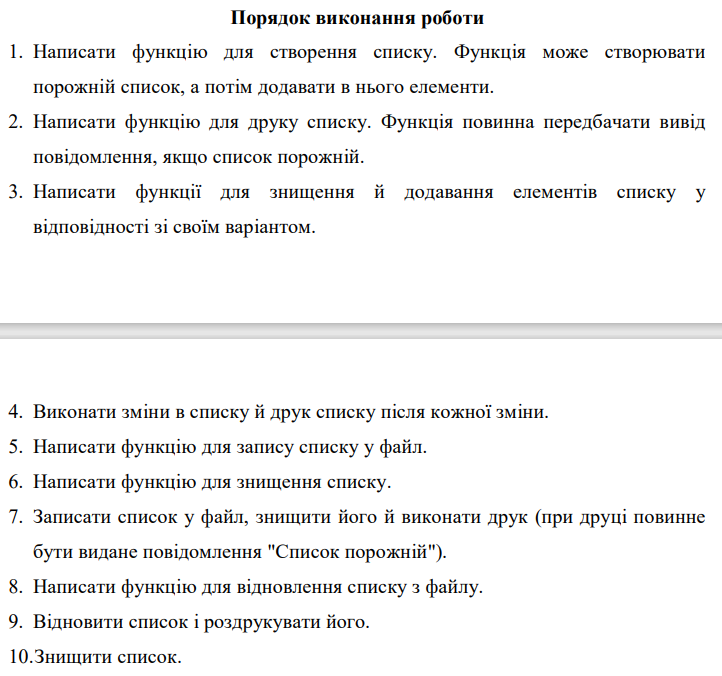
Статус: Ознайомлений

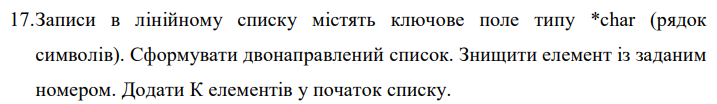
**Виконання роботи:**

* 1. **Опрацювання завдання та вимог до програм та середовища:**
* Epic 6 Task 2 - Requirements management (understand tasks) and design activities (draw flow diagrams and estimate tasks 3-7)

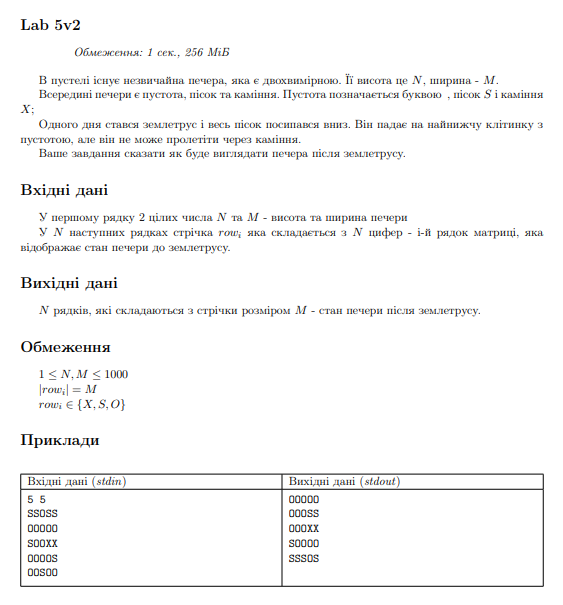
Створити блок-схему для 1 з програмного коду

* Task 3 - Lab# programming: VNS Lab 10

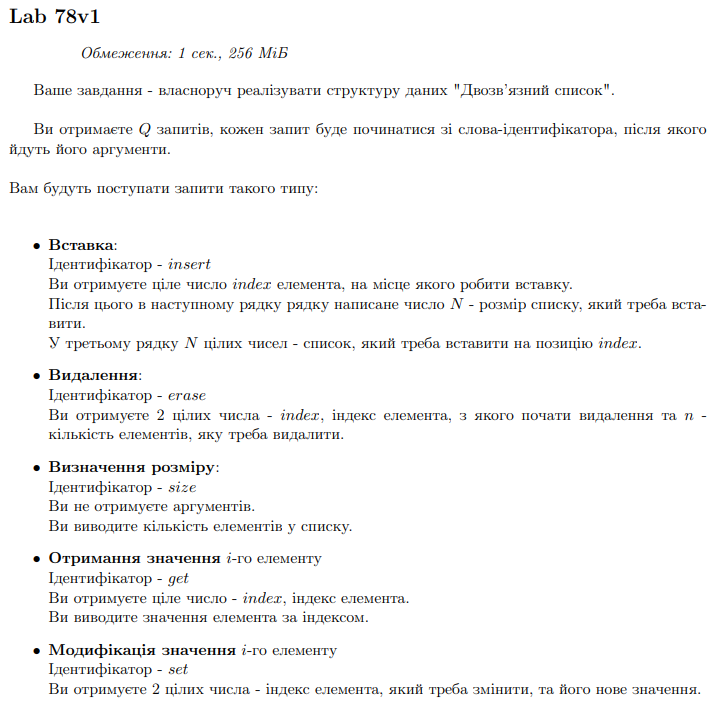


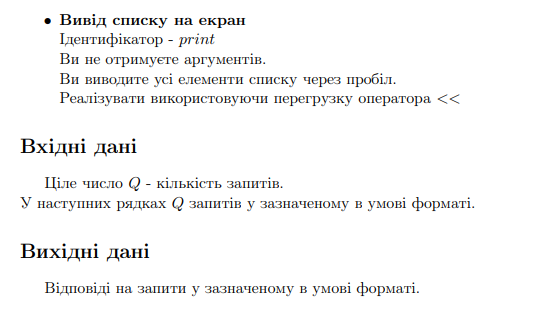


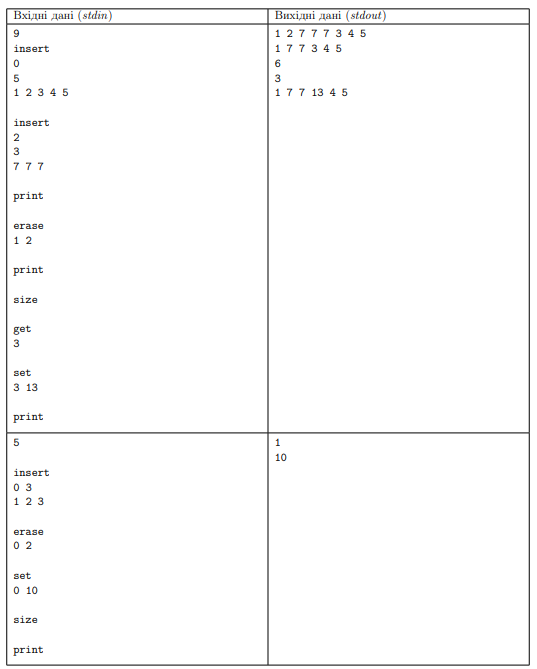
* Task 4 - Lab# programming: Algotester Lab 5



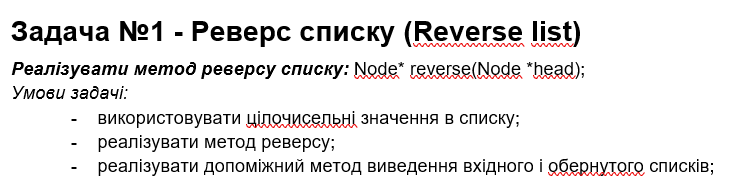
* Task 5 - Lab# programming: Algotester Lab 7-8

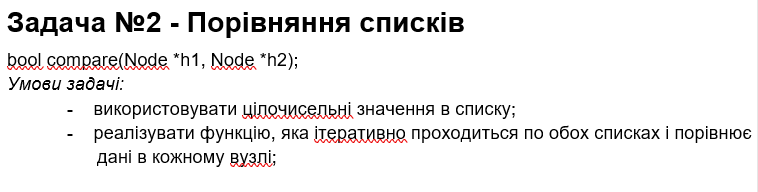


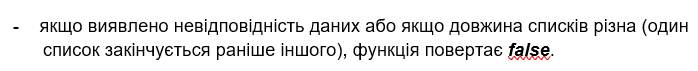


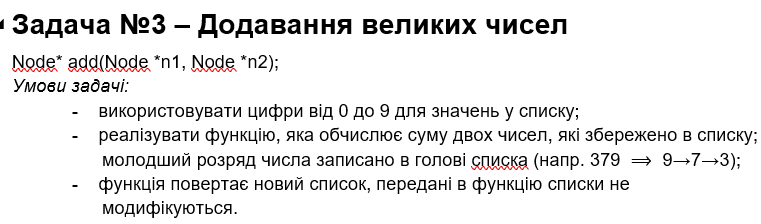


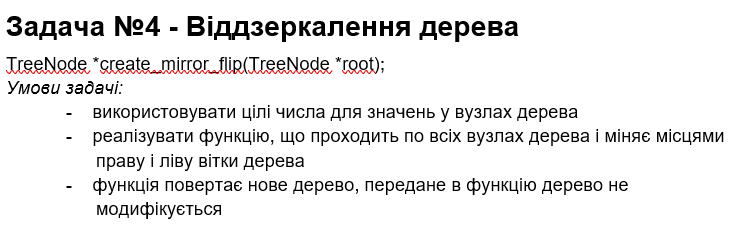
* Task 6 - Practice# programming: Class Practice Task

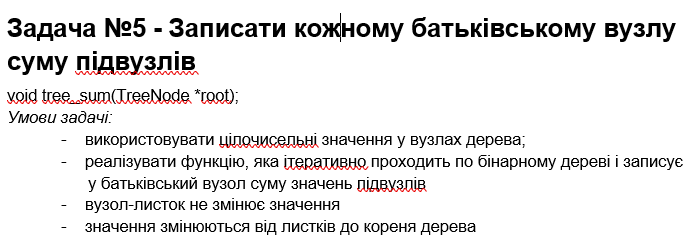




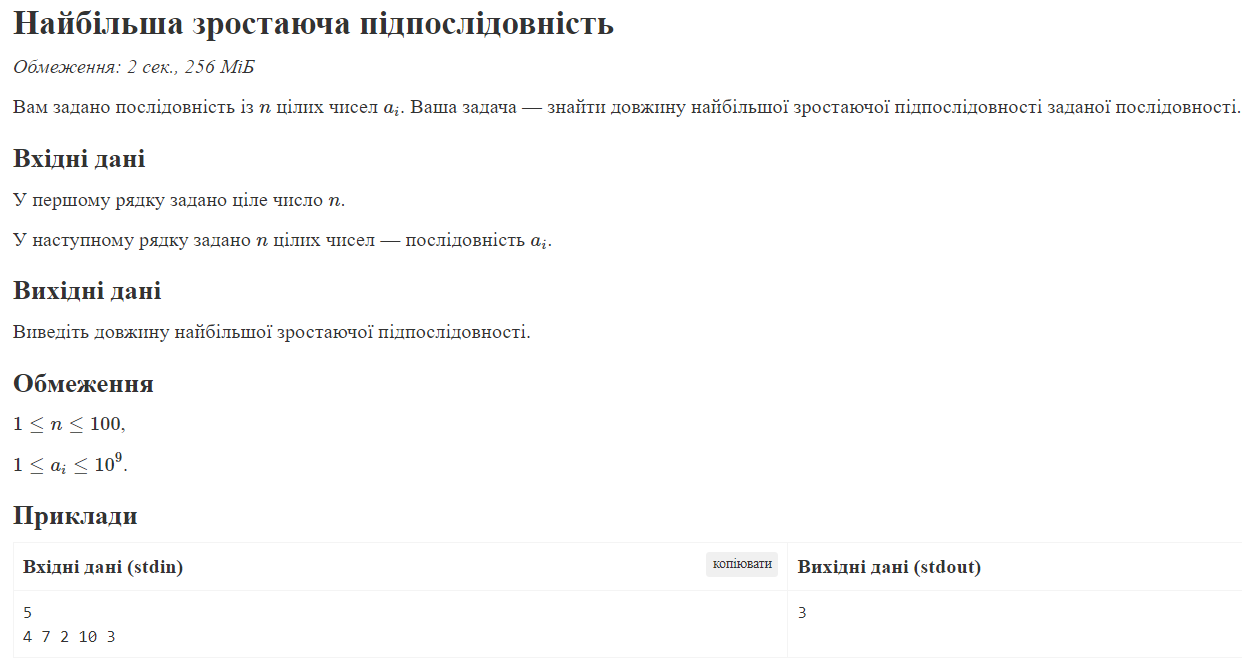








* Task 7 - Practice# programming: Self Practice Task



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

Блок-схема

* Task 7 - Practice# programming: Self Practice Task

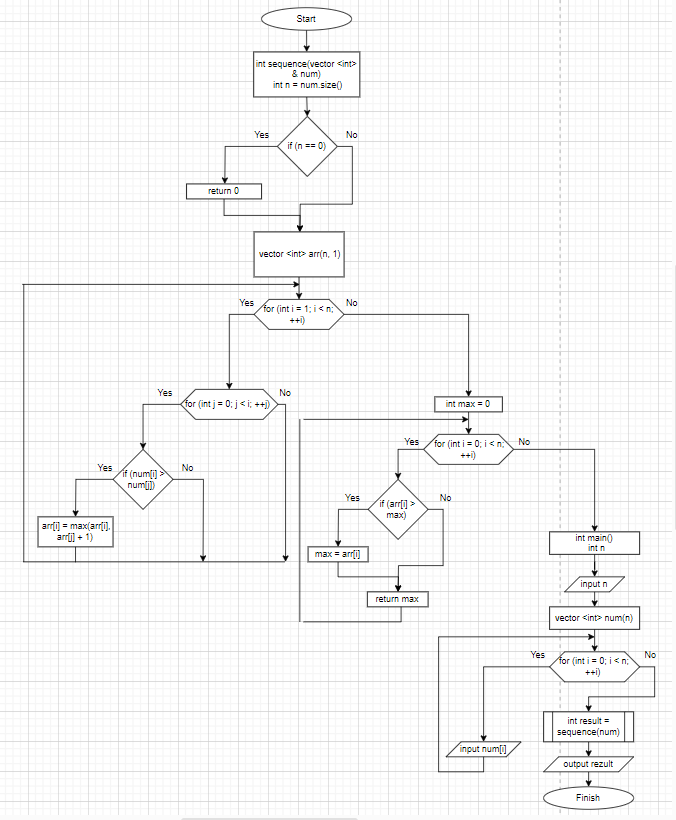


Рисунок : Блок схема до програми №1

* 1. **Код програм з посиланням на зовнішні ресурси:**
* Task 4 - Lab# programming: Algotester Lab 5

1. #include <iostream>
2. #include <vector>
3. using namespace std;
4. int main()
5. {
6. int N, M; // Зчитування розмірів печери
7. cin >> N;
8. cin >> M;
9. vector <vector <char>> cave (N, vector <char> (M)); // Ініціалізація двовимірного вектора для представлення печери
10. for (int i = 0; i < N; i++) // Зчитування конфігурації печери
11. {
12. for (int j = 0; j < M; j++)
13. {
14. cin >> cave[i][j];
15. }
16. }
17. for (int j = 0; j < M; j++) // Зсув піску униз, якщо під ними є пустий простір
18. {
19. for (int i = N - 1; i >= 0; i--)
20. {
21. if (i + 1 < N)
22. {
23. if (cave[i][j] == 'S' && cave[i + 1][j] == 'O') // Перевірка, чи можна зсунути пісок вниз
24. {
25. swap(cave[i][j], cave[i + 1][j]);
26. i = N - 1; // Переведення індексу в кінцевий стан для перевірки відповідних елементів
27. }
28. }
29. }
30. }
31. cout << endl;
32. for (int i = 0; i < N; ++i) // Виведення нового стану печери
33. {
34. for (int j = 0; j < M; ++j)
35. {
36. cout << cave[i][j];
37. }
38. cout << endl;
39. }
40. return 0;
41. }

* Task 5 - Lab# programming: Algotester Lab 7-8

1. #include <iostream>
2. #include <list>
3. using namespace std;
4. int main()
5. {
6. list <int> myList; // Оголошення та ініціалізація двозв'язного списку
7. int Q; // Оголошення та зчитування кількості операцій
8. cin >> Q;
9. while (Q--)
10. {
11. string command; // Зчитування команди для кожної операції
12. cin >> command;
13. if (command == "insert")
14. {
15. int index, N; // Вставка елементів у вказану позицію
16. cin >> index;
17. cin >> N;
18. for (int i = 0; i < N; ++i)
19. {
20. int value;
21. cin >> value;
22. auto it = myList.begin(); // Знаходження позиції для вставки та виконання вставки
23. advance(it, index + i);
24. myList.insert(it, value);
25. }
26. }
27. else if (command == "erase")
28. {
29. int index, n; // Видалення певної кількості елементів з початкової позиції
30. cin >> index >> n;
31. auto start = myList.begin();
32. advance(start, index);
33. auto end = start;
34. advance(end, n);
35. myList.erase(start, end);
36. }
37. else if (command == "size")
38. {
39. cout << myList.size() << endl; // Виведення розміру списку
40. }
41. else if (command == "get")
42. {
43. int index; // Отримання та виведення елемента за певною позицією
44. cin >> index;
45. auto it = myList.begin();
46. advance(it, index);
47. cout << \*it << endl;
48. }
49. else if (command == "set")
50. {
51. int index, value;  // Зміна значення елемента за певною позицією
52. cin >> index;
53. cin >> value;
54. auto it = myList.begin();
55. advance(it, index);
56. \*it = value;
57. }
58. else if (command == "print")
59. {
60. for (const auto& element : myList) // Виведення всіх елементів списку
61. {
62. cout << element << " ";
63. }
64. cout << endl;
65. }
66. }
67. return 0;
68. }

* Task 7 - Practice# programming: Self Practice Task

1. #include <iostream>
2. #include <vector>
3. using namespace std;
4. int sequence(vector <int> & num) //довжина найбільшої зростаючої підпослідовності
5. {
6. int n = num.size();
7. if (n == 0) // Якщо вхідна послідовність порожня, найбільша зростаюча підпослідовність також порожня
8. {
9. return 0;
10. }
11. vector <int> arr(n, 1); // зберігання довжини найбільшої зростаючої підпослідовності
12. for (int i = 1; i < n; ++i)
13. {
14. for (int j = 0; j < i; ++j)
15. {
16. if (num[i] > num[j])
17. {
18. arr[i] = max(arr[i], arr[j] + 1);
19. }
20. }
21. }
22. int max = 0;
23. for (int i = 0; i < n; ++i)
24. {
25. if (arr[i] > max)
26. {
27. max = arr[i];
28. }
29. }
30. return max;
31. }
32. int main()
33. {
34. int n;
35. cin >> n; // Зчитування розміру послідовності
36. vector <int> num(n);
37. for (int i = 0; i < n; ++i) // Зчитування елементів послідовності
38. {
39. cin >> num[i];
40. }
41. int result = sequence(num);
42. cout << result << endl;
43. return 0;
44. }

* Task 6 - Practice# programming: Class Practice Task

1.

1. #include <iostream>
2. using namespace std;
3. class Node // Клас для представлення вузла списку
4. {
5. public:
6. int data; // Значення вузла
7. Node\* next; // Вказівник на наступний вузол
8. Node(int value) : data(value), next(nullptr) {} // Конструктор для ініціалізації значення вузла
9. };
10. class LinkedList // Клас для представлення зв'язаного списку
11. {
12. private:
13. Node\* head; // Вказівник на голову списку
14. public:
15. LinkedList() : head(nullptr) {} // Конструктор для ініціалізації голови списку
16. void insert(int value) // Вставлення нового вузла на початок списку
17. {
18. Node\* newNode = new Node(value);
19. newNode->next = head;
20. head = newNode;
21. }
22. Node\* reverse(Node\* head) // Обертання списку
23. {
24. Node\* prev = nullptr;
25. Node\* current = head;
26. Node\* next = nullptr;
27. while (current != nullptr)
28. {
29. next = current->next;
30. current->next = prev;
31. prev = current;
32. current = next;
33. }
34. return prev;
35. }
36. void print\_List(Node\* head) // Виведення елементів списку
37. {
38. Node\* current = head;
39. while (current != nullptr)
40. {
41. cout << current->data << " ";
42. current = current->next;
43. }
44. cout << "\n";
45. }
46. void print\_original() // Виведення оригінального та обернутого списку
47. {
48. cout << "Original List: ";
49. print\_List(head);
50. Node\* reversedHead = reverse(head);
51. cout << "Reversed List: ";
52. print\_List(reversedHead);
53. }
54. };
55. int main()
56. {
57. LinkedList myList;
58. myList.insert(1); // Вставка елементів до списку
59. myList.insert(2);
60. myList.insert(3);
61. myList.insert(4);
62. myList.print\_original(); // Виведення оригінального та обернутого списку
63. return 0;
64. }

2.

#include <iostream>

using namespace std;

class Node // Клас для представлення вузла списку

{

public:

    int data; // Значення вузла

    Node\* next; // Вказівник на наступний вузол

    Node(int value) : data(value), next(nullptr) {} // Конструктор для ініціалізації значення вузла

};

bool compare(Node\* h1, Node\* h2) // Порівняння двох списків

{

    while (h1 != nullptr && h2 != nullptr)

    {

        if (h1->data != h2->data)

        {

            return false;  // Знайдена різниця в даних вузлів

        }

        h1 = h1->next;

        h2 = h2->next;

    }

    // Перевірка чи обидва списки закінчилися

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

}

int main()

{

    // Приклад використання

    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);

    // Виведення результату порівняння

    if (compare(list1, list2))

    {

        cout << "Списки рівні\n";

    }

    else

    {

        cout << "Списки відрізняються\n";

    }

    // 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;

    }

    return 0;

}

3.

#include <iostream>

using namespace std;

class Node // Клас для представлення вузла зв'язаного списку

{

public:

    int data; // Значення вузла

    Node\* next; // Вказівник на наступний вузол

    Node(int value) : data(value), next(nullptr) {} // Конструктор для ініціалізації значення вузла

};

Node\* add(Node\* n1, Node\* n2) // Функція для додавання двох чисел, представлених у вигляді списків

{

    Node\* result = nullptr;

    Node\* current = nullptr;

    int carry = 0;

    while (n1 != nullptr || n2 != nullptr || carry != 0) // Проходимо обидва списки та додаємо числа разом з переносом

    {

        int sum = (n1 ? n1->data : 0) + (n2 ? n2->data : 0) + carry;

        carry = sum / 10;

        Node\* newNode = new Node(sum % 10); // Створюємо новий вузол з отриманою сумою

        if (result == nullptr) // Додаємо вузол до результуючого списку

        {

            result = newNode;

            current = result;

        }

        else

        {

            current->next = newNode;

            current = current->next;

        }

        if (n1) n1 = n1->next; // Переходимо до наступного вузла в обох вхідних списках

        if (n2) n2 = n2->next;

    }

    return result;

}

void printList(Node\* head) // Функція для виведення списку на екран

{

    Node\* current = head;

    while (current != nullptr)

    {

        cout << current->data;

        current = current->next;

        if (current != nullptr)

        {

            cout << " -> ";

        }

    }

    cout << "\n";

}

int main()

{

    Node\* num1 = new Node(7);

    num1->next = new Node(2);

    num1->next->next = new Node(4);

    Node\* num2 = new Node(5);

    num2->next = new Node(6);

    num2->next->next = new Node(4);

    cout << "Число 1\n";

    printList(num1);

    cout << "\nЧисло 2\n";

    printList(num2);

    Node\* sum = add(num1, num2);

    cout << "\nСума\n";

    printList(sum);

    while (num1 != nullptr) // Звільнення пам'яті

    {

        Node\* temp = num1;

        num1 = num1->next;

        delete temp;

    }

    while (num2 != nullptr)

    {

        Node\* temp = num2;

        num2 = num2->next;

        delete temp;

    }

    while (sum != nullptr)

    {

        Node\* temp = sum;

        sum = sum->next;

        delete temp;

    }

    return 0;

}

4.

#include <iostream>

using namespace std;

class TreeNode // Клас, що представляє вузол бінарного дерева

{

public:

    int data;

    TreeNode\* left;

    TreeNode\* right;

    TreeNode(int value) : data(value), left(nullptr), right(nullptr) {} // Конструктор для ініціалізації значень вузла

};

void printNodes(TreeNode\* root)  // Функція для виведення інформації про вузол та його дочірні вузли

{

    if (root == nullptr)

    {

        return;

    }

    cout << "Node " << root->data << ":\n";

    if (root->left)

    {

        cout << "  Node " << root->left->data << " (Left child): " << "\n";

    }

    if (root->right)

    {

        cout << "  Node " << root->right->data << " (Right child): " << "\n";

    }

}

void deleteSubtree(TreeNode\* root) // Функція для видалення піддерева

{

    if (root == nullptr)

    {

        return;

    }

    deleteSubtree(root->left);// Рекурсивно звільнюємо пам'ять для всіх вузлів дерева

    deleteSubtree(root->right);

    delete root; // Звільнюємо пам'ять для поточного вузла

}

TreeNode\* copySubtree(TreeNode\* root) // Функція для копіювання піддерева

{

    if (root == nullptr)

    {

        return nullptr;

    }

    TreeNode\* newRoot = new TreeNode(root->data); // Створюємо новий вузол з таким самим значенням

    newRoot->left = copySubtree(root->left); // Рекурсивно копіюємо ліву та праву вітки

    newRoot->right = copySubtree(root->right);

    return newRoot;

}

TreeNode\* create\_mirror\_tree(TreeNode\* root) // Функція для створення дзеркального відображення дерева

{

    if (root == nullptr)

    {

        return nullptr;

    }

    TreeNode\* newRoot = new TreeNode(root->data);  // Створюємо новий вузол з таким самим значенням

    newRoot->right = create\_mirror\_tree(root->left); // Рекурсивно створюємо дзеркальні вітки (праву і ліву)

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

    return newRoot;

}

int main()

{

    // Створюємо оригінальне дерево

    TreeNode\* root = new TreeNode(1);

    root->left = new TreeNode(2);

    root->right = new TreeNode(3);

    root->left->left = new TreeNode(4);

    root->left->right = new TreeNode(5);

    cout << "Оригінальне дерево:\n";

    printNodes(root);

    TreeNode\* mirroredRoot = create\_mirror\_tree(root);  // Створюємо дзеркальне відображення дерева

    cout << "Дзеркальне відображення дерева:\n"; // Виводимо дзеркальне відображення дерева

    printNodes(mirroredRoot);

    deleteSubtree(root); // Звільнюємо пам'ять оригінального дерева та його дзеркального відображення

    deleteSubtree(mirroredRoot);

    return 0;

}

5.

#include <iostream>

using namespace std;

class TreeNode // Клас, що представляє вузол бінарного дерева

{

public:

    int data;

    TreeNode\* left;

    TreeNode\* right;

    TreeNode(int value) : data(value), left(nullptr), right(nullptr) {} // Конструктор для ініціалізації значень вузла

};

void print\_tree(TreeNode\* root) // Функція для виведення дерева

{

    if (root == nullptr)

    {

        return;

    }

    cout << "Node " << root->data << ":\n";

    if (root->left)

    {

        cout << "  Left child: " << root->left->data << "\n";

    }

    if (root->right)

    {

        cout << "  Right child: " << root->right->data << "\n";

    }

    if (root->left || root->right)

    {

        print\_tree(root->left); // Якщо вузол не є листком, виводимо його дітей

        print\_tree(root->right);

    }

}

void deleteSubtree(TreeNode\* root) // Функція для видалення піддерева

{

    if (root == nullptr)

    {

        return;

    }

    deleteSubtree(root->left); // Рекурсивно звільнюємо пам'ять для всіх вузлів дерева

    deleteSubtree(root->right);

    delete root; // Звільнюємо пам'ять для поточного вузла

}

void tree\_sum(TreeNode\* root) // Функція для запису суми підвузлів в батьківський вузол

{

    if (root == nullptr)

    {

        return;

    }

    tree\_sum(root->left); // Рекурсивно обчислюємо суми для лівого і правого піддерева

    tree\_sum(root->right);

    if (root->left || root->right) // Обчислюємо суму підвузлів і записуємо її в батьківський вузол (лише якщо він не є листком)

    {

        int sum = 0;

        if (root->left)

        {

            sum += root->left->data;

        }

        if (root->right)

        {

            sum += root->right->data;

        }

        root->data = sum;

    }

}

int main()

{

    // Створюємо бінарне дерево

    TreeNode\* root = new TreeNode(3);

    root->left = new TreeNode(6);

    root->right = new TreeNode(7);

    root->left->left = new TreeNode(17);

    root->left->right = new TreeNode(19);

    cout << "До виконання функції tree\_sum:\n";

    print\_tree(root);

    tree\_sum(root);

    cout << "\nПісля виконання функції tree\_sum:\n";

    print\_tree(root);

    deleteSubtree(root);

    return 0;

}

* Task 3 - Lab# programming: VNS Lab 10

#include <iostream>

#include <fstream>

#include <cstring>

using namespace std;

struct Node // Структура для представлення вузла списку

{

    char\* data; // Ключове поле - рядок символів

    Node\* next; // Вказівник на наступний вузол

    Node\* prev; // Вказівник на попередній вузол

};

class DoublyLinkedList // Клас для реалізації двонаправленого списку

{

private:

    Node\* head; // Вказівник на початок списку

public:

    DoublyLinkedList() : head(nullptr) {}

    void createList() // Функція для створення порожнього списку

    {

        head = nullptr;

        cout << "Створено порожній список\n";

    }

    void addElement(const char\* newData) // Функція для додавання елемента в початок списку

    {

        Node\* newNode = new Node; // Створення нового вузла

        newNode->data = strdup(newData); // Копіювання рядка символів

        newNode->next = nullptr;

        newNode->prev = nullptr;

        if (head == nullptr)

        {

            head = newNode;

        }

        else

        {

            newNode->next = head;

            head->prev = newNode;

            head = newNode;

        }

        cout << "Додано елемент: " << newData << endl;

        printList();

    }

    void deleteElement(int position) // Функція для видалення елемента за заданим номером

    {

        if (head == nullptr)

        {

            cout << "Список порожній. Нема що видаляти\n";

            return;

        }

        Node\* current = head;

        int count = 1;

        while (current != nullptr && count < position)

        {

            current = current->next;

            count++;

        }

        if (current == nullptr)

        {

            cout << "Елемент не знайдено\n";

            return;

        }

        if (current->prev != nullptr)

        {

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

        }

        else

        {

            head = current->next;

        }

        if (current->next != nullptr)

        {

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

        }

        cout << "Елемент видалено. Новий вигляд списку:\n";

        printList();

    }

    void printList() // Функція для друку всього списку

    {

        if (head == nullptr)

        {

            cout << "Список порожній\n";

            return;

        }

        Node\* current = head;

        cout << "Список: ";

        while (current != nullptr)

        {

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

            current = current->next;

        }

        cout << endl;

    }

    void writeToFile(const char\* filename)  // Функція для запису списку у файл

    {

        ofstream outFile(filename);

        Node\* current = head;

        while (current != nullptr)

        {

            outFile << current->data << endl;

            current = current->next;

        }

        outFile.close();

        cout << "Список записано у файл: " << filename << endl;

    }

    void destroyList() // Функція для знищення всього списку

    {

        Node\* current = head;

        while (current != nullptr)

        {

            Node\* nextNode = current->next;

            delete[] current->data;

            delete current;

            current = nextNode;

        }

        head = nullptr;

        cout << "Список знищено\n";

    }

    void restoreFromFile(const char\* filename)  // Функція для відновлення списку з файлу

    {

        ifstream inFile(filename);

        if (!inFile.is\_open())

        {

            cout << "Помилка відкриття файлу\n";

            return;

        }

        destroyList(); // Очищення поточного списку перед відновленням

        char buffer[256];

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

        {

            addElement(buffer);

        }

        inFile.close();

        cout << "Список відновлено\n";

    }

};

int main()

{

    DoublyLinkedList myList; // Створення об'єкта класу

    myList.createList(); // Створення порожнього списку

    myList.addElement("Елемент1");

    myList.addElement("Елемент2");

    myList.addElement("Елемент3");

    myList.deleteElement(2);

    myList.addElement("Елемент4");

    myList.addElement("Елемент5");

    myList.writeToFile("output.txt");

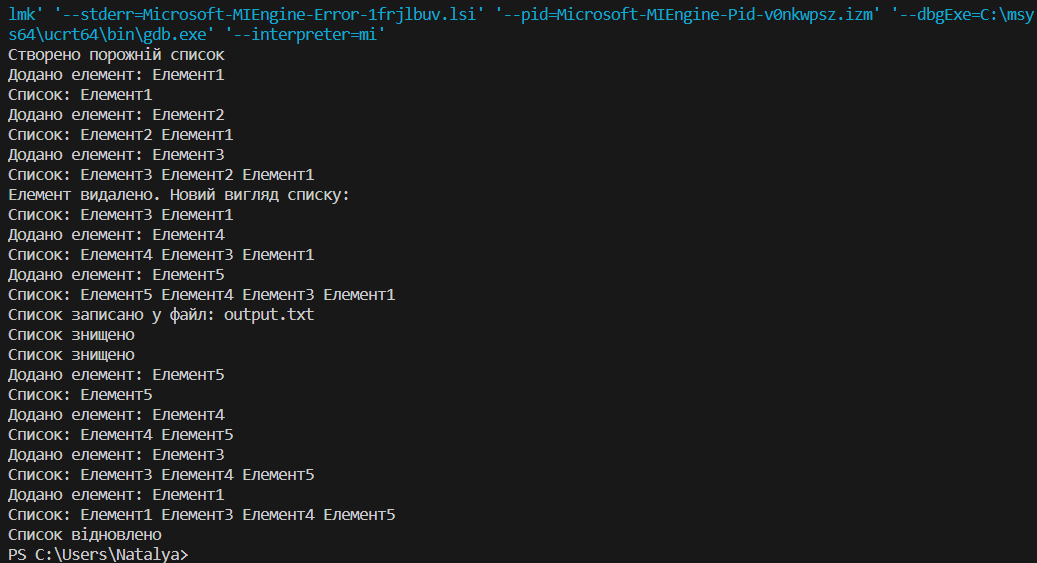
    myList.destroyList();

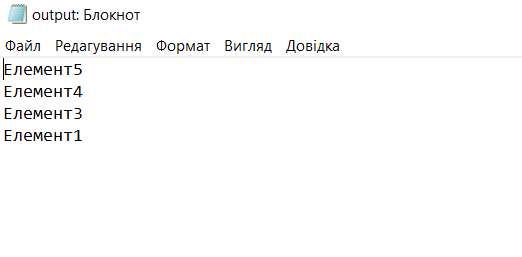
    myList.restoreFromFile("output.txt");

    return 0;

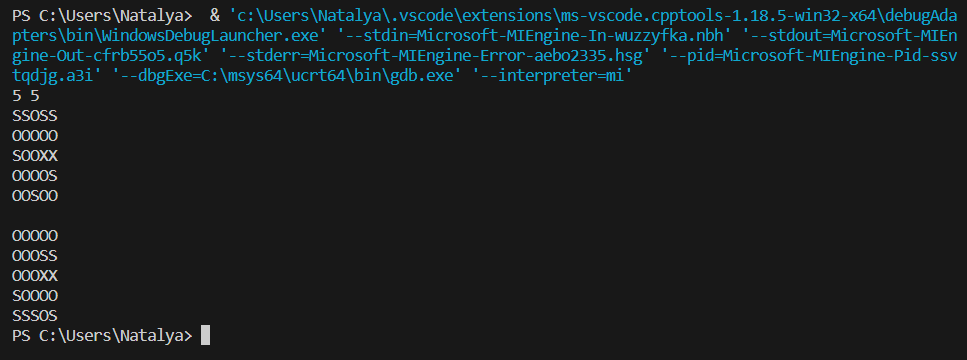
}

* + 1. **Результати виконання завдань, тестування та фактично затрачений час:**
* Task 3 - Lab# programming: VNS Lab 10

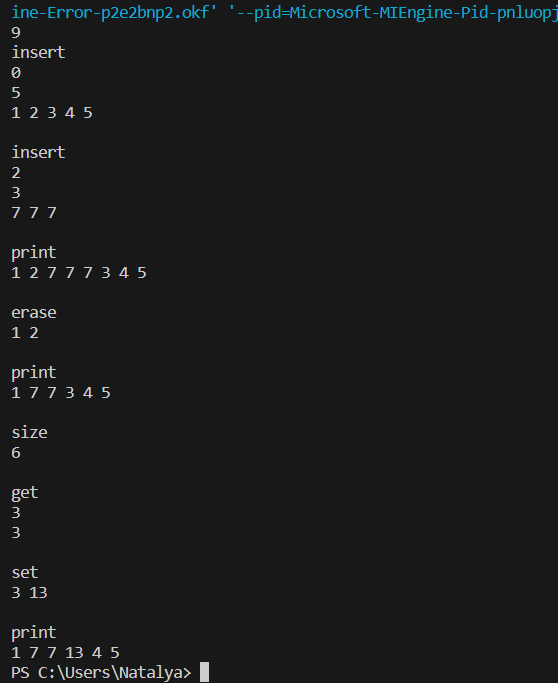


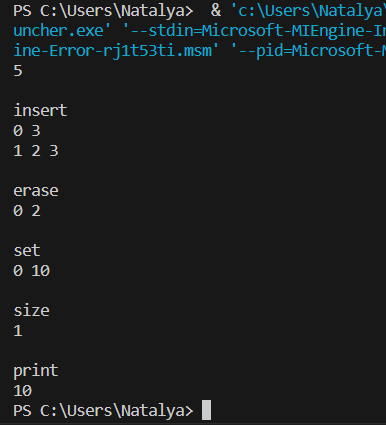


* Task 4 - Lab# programming: Algotester Lab 5



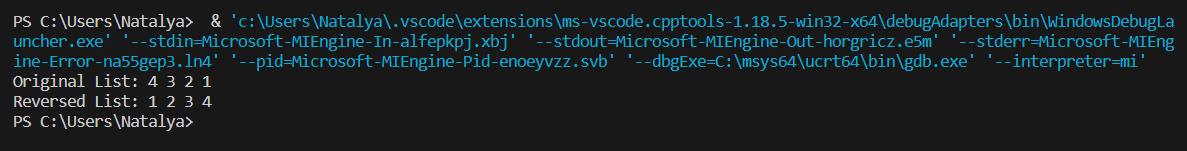
* Task 5 - Lab# programming: Algotester Lab 7-8



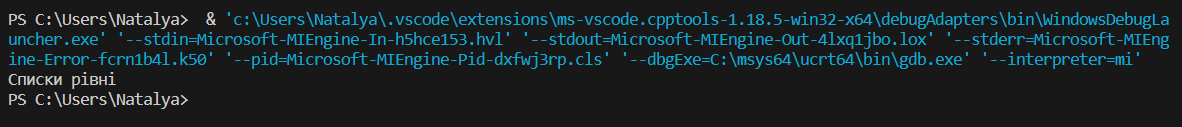


* Task 6 - Practice# programming: Class Practice Task

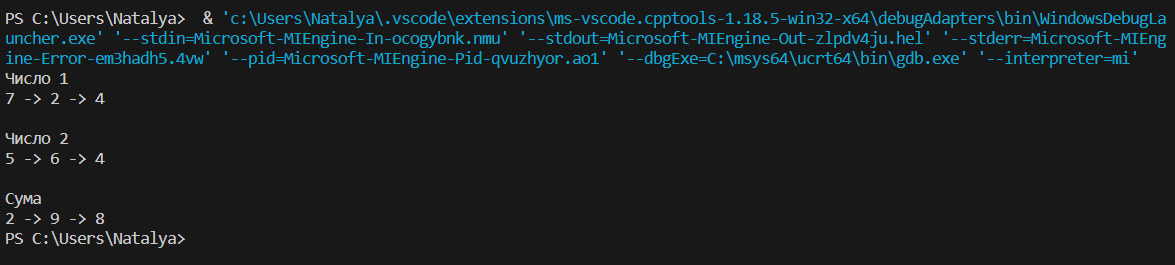
1.



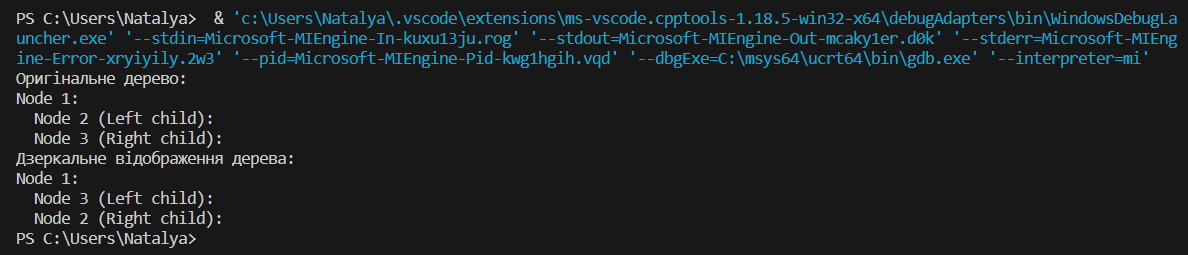
2.



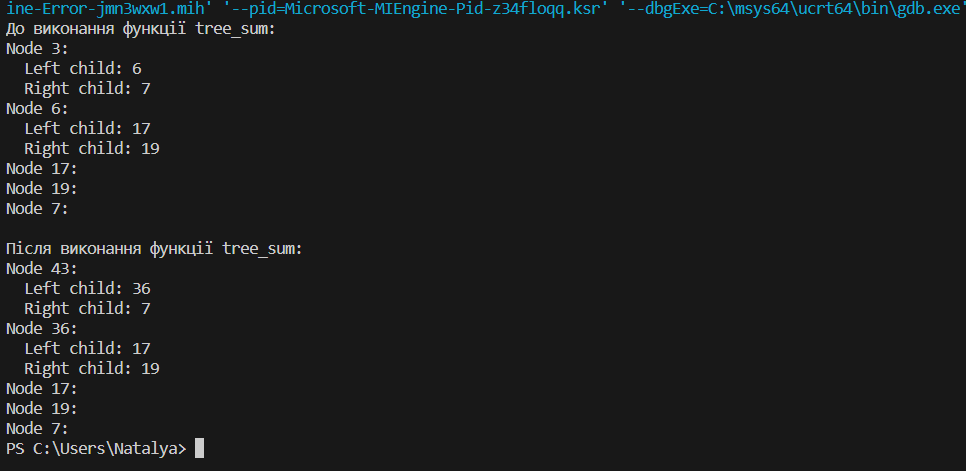
3.



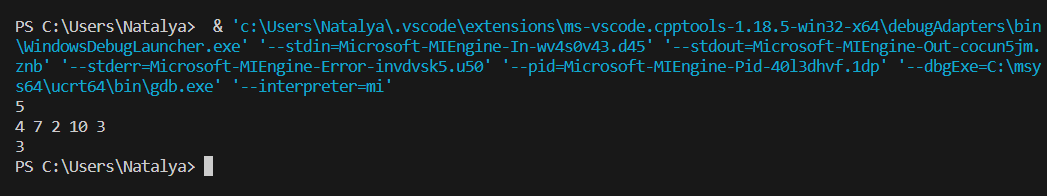
4.



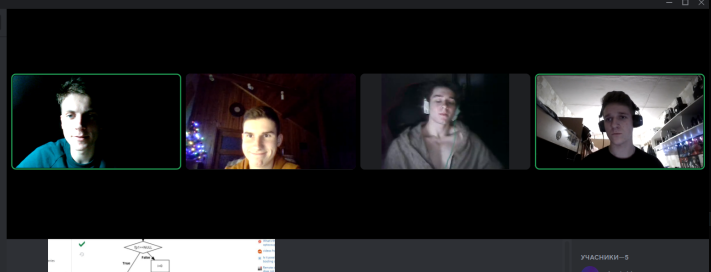
5.

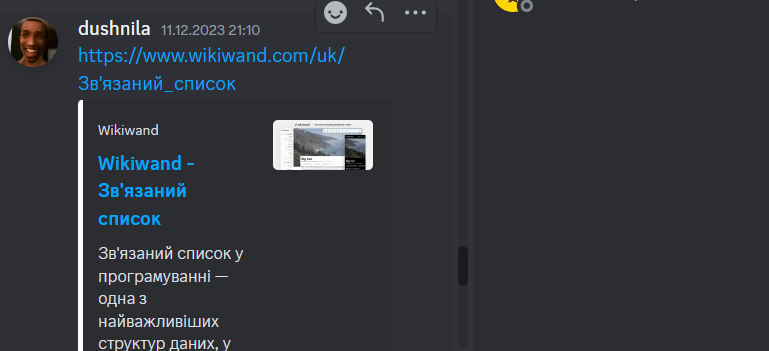


* Task 7 - Practice# programming: Self Practice Task

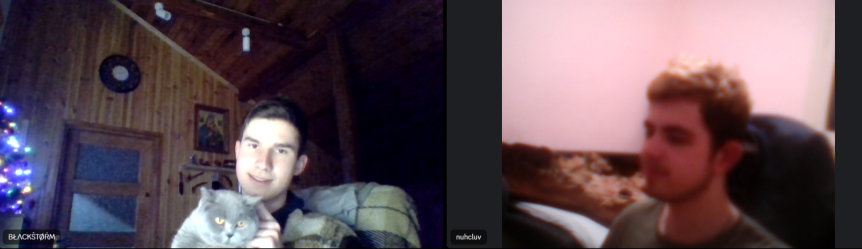


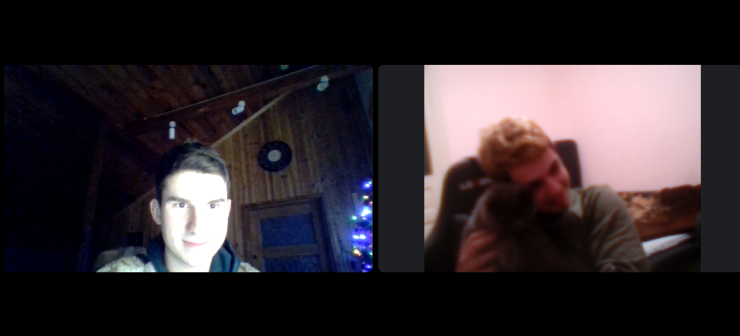
**4. Зустріч з командою:**

****

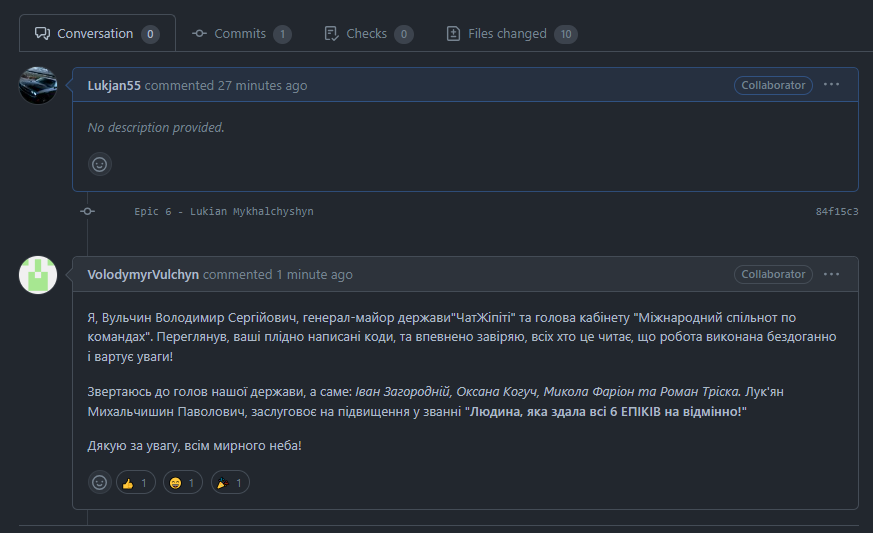
****

**PS: кооперація з іншими командами, навіть коти беруть участь в написанні епіків**

****

****

**Коментар від одногрупників в GitHub:**

****

**Посилання на pull request**

[**https://github.com/artificial-intelligence-department/ai\_programming\_playground/pull/1099**](https://github.com/artificial-intelligence-department/ai_programming_playground/pull/1099)

**Висновки:** Я навчився використовувати стеки, класи, структури для реалізації коду під ті задачі, які поставали переді мною. Також взнав що існують дерева, та для чого вони взагалі потрібні. Навчився і кооперуватись з іншими командами, для вирішування спільних проблем, та і взагалі допомагати один-одному, в загальному була і цікава задача про печеру, пісок, і каміння – нагадала старий добрий тетріс.