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: «Основи програмування»

до:

ВНС Лабораторної Роботи № 10
Алготестер Лабораторної Роботи № 5
Алготестер Лабораторної Роботи № 7-8
Практичних Робіт до блоку № 6

Виконав:
Студент групи ШІ-12
Сирватка Олександр Ігорович

Львів 2024

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

Мета роботи: навчитись працювати з динамічними структурами, спробувати написати власні алгоритми для таких структур як: черга, стек, список та дерево.

Теоретичні відомості:

1) Структури
2) Класи
3) Список
4) Подвійний список
5) Бінарне дерево

Індивідуальний план опрацювання теорії:

- Тема №1 Структури (50 хв) (https://www.youtube.com/watch?v=999IE-6b7_s)
- Тема №2 Класи (50 хв) (https://www.youtube.com/watchv=ZbsukxxV5_Q&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=95)
- Тема №3 Список (70 хв) (https://www.youtube.com/watchv=-25REjF_atI&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=141)
- Тема №4 Двозв’язний список (40 хв) (https://www.youtube.com/watch?v=QLzu2_QFoE&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g)
- Тема №5 Бінарне дерево (50 хв) (https://www.youtube.com/watch?v=qBFzNW0ALxQ&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g)

Виконання роботи

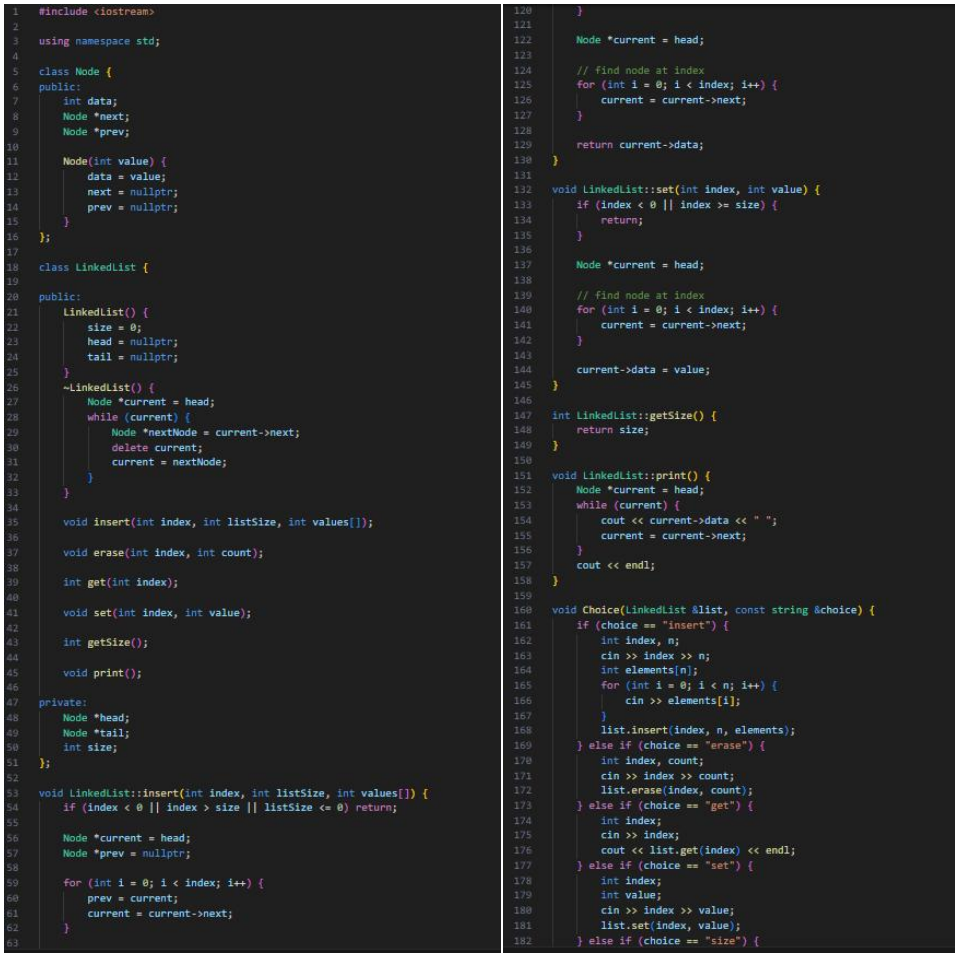
Завдання №1 Epic 6 Task 3 - Lab# programming: VNS Lab 10

```cpp
#include <iostream>
#include <fstream>

using namespace std;

struct Node {
    int key;
    Node* next;
};

// Функція для створення порожнього списку
Node* createList() {
    return nullptr;  // Створюємо порожній список
}

// Функція для друку списку
void printList(Node* head) {
    if (!head) {
        cout << "Список порожній" << endl;
        return;
    }
    Node* temp = head;
    while (temp) {
        cout << temp->key << " ";
        temp = temp->next;
    }
    cout << endl;
}

// Функція для додавання елемента в список
void addElement(Node*& head, int value, int position) {
    Node* newNode = new Node{value, nullptr};  // Створюємо новий вузол
    if (position == 0 || !head) {  // Додавання на початок списку
        newNode->next = head;  // Новий вузол вказує на поточну голову
        head = newNode;  // Оновлюємо голову списку
        return;
    }

    Node* temp = head;  // Починаємо з голови
    for (int i = 0; i < position - 1 && temp->next; ++i) {
        temp = temp->next;  // Переходимо до вузла перед позицією вставки
    }
    newNode->next = temp->next;  // Новий вузол вказує на наступний вузол
    temp->next = newNode;  // Поточний вузол вказує на новий вузол
}

// Функція для видалення елемента зі списку
void deleteElement(Node*& head, int position) {
    if (!head) return;

    Node* temp = head;
    if (position == 0) {
        head = head->next;
        delete temp;
        return;
    }

    for (int i = 0; temp && i < position - 1; ++i) {
        temp = temp->next;
    }
    if (!temp || !temp->next) return;

    Node* next = temp->next->next;
    delete temp->next;
    temp->next = next;
}

// Функція для запису списку у файл
void writeToFile(Node* head, const string& filename) {
    ofstream file(filename);

    // Перевірка, чи вдалося відкрити файл
    if (!file.is_open()) {
        cerr << "Не вдалося відкрити файл для запису: " << filename << endl;
        return;
    }

    Node* temp = head;
    while (temp) {
        file << temp->key << endl;
        temp = temp->next;
    }
    file.close();  // Закриваємо файл після запису
}

// Функція для відновлення списку з файлу
Node* readFromFile(const string& filename) {
    ifstream file(filename);

    // Перевірка, чи вдалося відкрити файл
    if (!file.is_open()) {
        cerr << "Не вдалося відкрити файл: " << filename << endl;
        return nullptr;
    }

    Node* head = nullptr;
    Node* tail = nullptr;
    int value;

    while (file >> value) {
        Node* newNode = new Node{value, nullptr};

        if (!head) {
            head = tail = newNode;  // Якщо список порожній, новий вузол стає головою
        } else {
            tail->next = newNode;  // Додаємо новий вузол в кінець списку
            tail = newNode;        // Оновлюємо tail
        }
    }

    file.close();  // Закриваємо файл після зчитування
    return head;   // Повертаємо голову списку
}

// Функція для знищення списку
void deleteList(Node*& head) {
    while (head) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
}

int main() {
    Node* list = createList();

    addElement(list, 10, 0);  // Додавання елементів
    addElement(list, 20, 1);
    addElement(list, 30, 2);
    printList(list);

    deleteElement(list, 1);  // Видалення елемента
    printList(list);

    writeToFile(list, "list.txt");  // Запис у файл
    deleteList(list);
    printList(list);  // Список порожній

    list = readFromFile("list.txt");  // Відновлення з файлу
    printList(list);

    deleteList(list);  // Очищення пам'яті
    return 0;
}
```

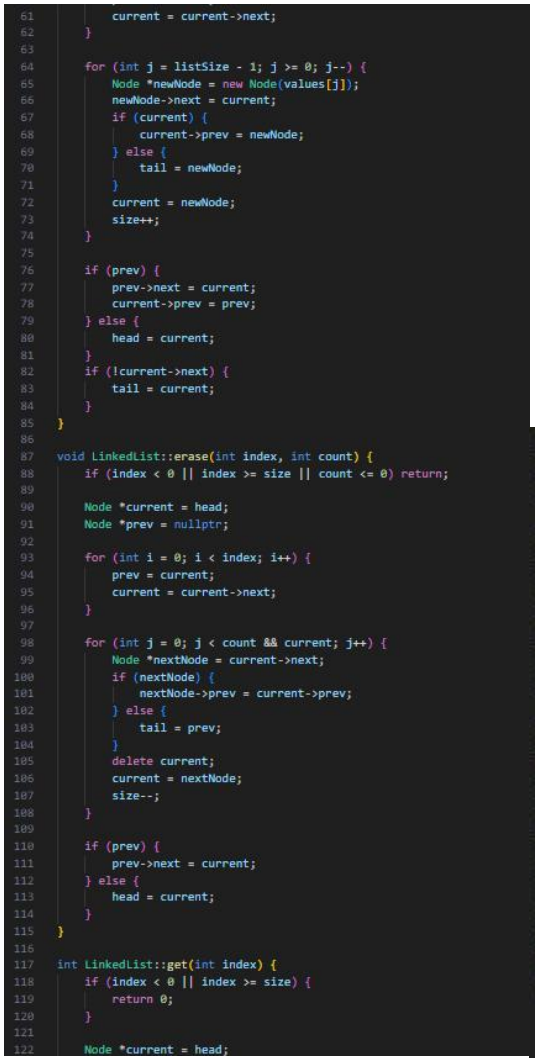
Завдання №2 Epic 6 Task 4 - Lab# programming: Algotester Lab 5

```cpp
#include <iostream>
#include <vector>
#include <string>

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 j = 0; j < M; ++j) {
        int bottom = N;

        // Обробляємо стовпець зверху вниз
        for (int i = N - 1; i >= 0; --i) {
            if (cave[i][j] == 'X') {
                bottom = i;
            } else if (cave[i][j] == 'S') {

                cave[i][j] = '0';
                cave[--bottom][j] = 'S';
            }
        }
    }


    for (int i = 0; i < N; ++i) {
        cout << cave[i] << endl;
    }

    return 0;
}

```

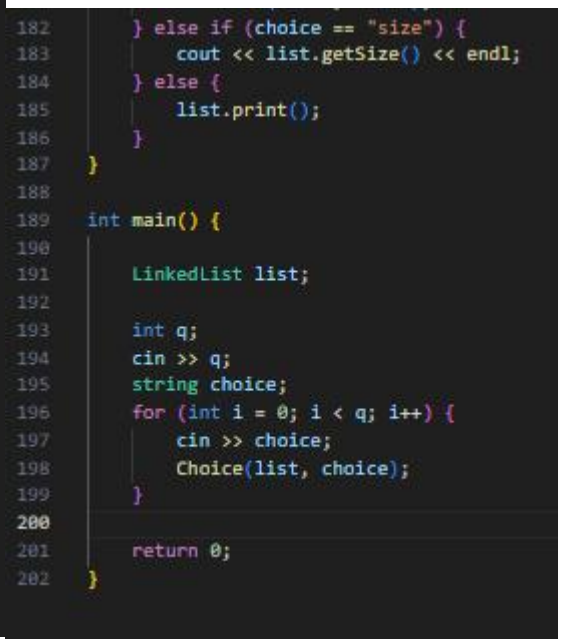
Завдання №3 Epic 6 Task 5 - Lab# programming: Algotester Lab 7-8 v1



```cpp
#include <iostream>

using namespace std;

class Node {
public:
    int data;
    Node *next;
    Node *prev;

    Node(int value) {
        data = value;
        next = nullptr;
        prev = nullptr;
    }
};

class LinkedList {

public:
    LinkedList() {
        size = 0;
        head = nullptr;
        tail = nullptr;
    }
    ~LinkedList() {
        Node *current = head;
        while (current) {
            Node *nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }

    void insert(int index, int listSize, int values[]);

    void erase(int index, int count);

    int get(int index);

    void set(int index, int value);

    int getSize();

    void print();

private:
    Node *head;
    Node *tail;
    int size;
};

void LinkedList::insert(int index, int listSize, int values[]) {
    if (index < 0 || index > size || listSize <= 0) return;

    Node *current = head;
    Node *prev = nullptr;

    for (int i = 0; i < index; i++) {
        prev = current;
        current = current->next;
    }
    }

    Node *current = head;

    // find node at index
    for (int i = 0; i < index; i++) {
        current = current->next;
    }

    return current->data;
}

void LinkedList::set(int index, int value) {
    if (index < 0 || index >= size) {
        return;
    }

    Node *current = head;

    // find node at index
    for (int i = 0; i < index; i++) {
        current = current->next;
    }

    current->data = value;
}

int LinkedList::getSize() {
    return size;
}

void LinkedList::print() {
    Node *current = head;
    while (current) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

void Choice(LinkedList &list, const string &choice) {
    if (choice == "insert") {
        int index, n;
        cin >> index >> n;
        int elements[n];
        for (int i = 0; i < n; i++) {
            cin >> elements[i];
        }
        list.insert(index, n, elements);
    } else if (choice == "erase") {
        int index, count;
        cin >> index >> count;
        list.erase(index, count);
    } else if (choice == "get") {
        int index;
        cin >> index;
        cout << list.get(index) << endl;
    } else if (choice == "set") {
        int index;
        int value;
        cin >> index >> value;
        list.set(index, value);
    } else if (choice == "size") {
```



```
        current = current->next;
    }

    for (int j = listSize - 1; j >= 0; j--) {
        Node *newNode = new Node(values[j]);
        newNode->next = current;
        if (current) {
            current->prev = newNode;
        } else {
            tail = newNode;
        }
        current = newNode;
        size++;
    }

    if (prev) {
        prev->next = current;
        current->prev = prev;
    } else {
        head = current;
    }
    if (!current->next) {
        tail = current;
    }
}

void LinkedList::erase(int index, int count) {
    if (index < 0 || index >= size || count <= 0) return;

    Node *current = head;
    Node *prev = nullptr;

    for (int i = 0; i < index; i++) {
        prev = current;
        current = current->next;
    }

    for (int j = 0; j < count && current; j++) {
        Node *nextNode = current->next;
        if (nextNode) {
            nextNode->prev = current->prev;
        } else {
            tail = prev;
        }
        delete current;
        current = nextNode;
        size--;
    }

    if (prev) {
        prev->next = current;
    } else {
        head = current;
    }
}

int LinkedList::get(int index) {
    if (index < 0 || index >= size) {
        return 0;
    }

    Node *current = head;
```



```
    } else if (choice == "size") {
        cout << list.getSize() << endl;
    } else {
        list.print();
    }
}

int main() {

    LinkedList list;

    int q;
    cin >> q;
    string choice;
    for (int i = 0; i < q; i++) {
        cin >> choice;
        Choice(list, choice);
    }

    return 0;
}
```

Завдання №3 Epic 6 Task 5 - Lab# programming: Algotester Lab 7-8 v2

```cpp
#include <iostream>

using namespace std;

template<typename T>
class Node {
public:
    T data;
    Node<T> *next;
    Node<T> *prev;

    Node(T value) {
        data = value;
        next = nullptr;
        prev = nullptr;
    }
};

template<typename T>
class LinkedList {

public:
    LinkedList() {
        size = 0;
        head = nullptr;
        tail = nullptr;
    }
    ~LinkedList() {
        Node<T> *current = head;
        while (current) {
            Node<T> *nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }

    void insert(int index, int listSize, T values[]);

    void erase(int index, int count);

    int get(int index);

    void set(int index, T value);

    int getSize();

    void print();

private:
    Node<T> *head;
    Node<T> *tail;
    int size;
};

template<class T>
void LinkedList<T>::insert(int index, int listSize, T values[]) {
    if (index < 0 || index > size || listSize <= 0) return;

    Node<T> *current = head;
    Node<T> *prev = nullptr;

    for (int i = 0; i < index; i++) {
        prev = current;
        current = current->next;
    }

    for (int j = listSize - 1; j >= 0; j--) {
        Node<T> *newNode = new Node<T>(values[j]);
        newNode->next = current;
        if (current) {
            current->prev = newNode;
        } else {
            tail = newNode;
        }
        current = newNode;
        size++;
    }

    if (prev) {
        prev->next = current;
        current->prev = prev;
    } else {
        head = current;
    }
    if (!current->next) {
        tail = current;
    }
}


template<class T>
void LinkedList<T>::erase(int index, int count) {
    if (index < 0 || index >= size || count <= 0) return;

    Node<T> *current = head;
    Node<T> *prev = nullptr;

    for (int i = 0; i < index; i++) {
        prev = current;
        current = current->next;
    }

    for (int j = 0; j < count && current; j++) {
        Node<T> *nextNode = current->next;
        if (nextNode) {
            nextNode->prev = current->prev;
        } else {
            tail = prev;
        }
        delete current;
        current = nextNode;
        size--;
    }

    if (prev) {
        prev->next = current;
    } else {
        head = current;
    }
}
```

```cpp
template<class T>
int LinkedList<T>::get(int index) {
    if (index < 0 || index >= size) {
        return 0;
    }

    Node<T> *current = head;

    for (int i = 0; i < index; i++) {
        current = current->next;
    }

    return current->data;
}

template<class T>
void LinkedList<T>::set(int index, T value) {
    if (index < 0 || index >= size) {
        return;
    }

    Node<T> *current = head;

    for (int i = 0; i < index; i++) {
        current = current->next;
    }
    current->data = value;
}

template<class T>
int LinkedList<T>::getSize() {
    return size;
}

template<class T>
void LinkedList<T>::print() {
    Node<T> *current = head;
    while (current) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

template<typename T>
void Choice(LinkedList<T> &list, const string &choice) {
    if (choice == "insert") {
        int index, n;
        cin >> index >> n;
        T elements[n];
        for (int i = 0; i < n; i++) {
            cin >> elements[i];
        }
        list.insert(index, n, elements);
    } else if (choice == "erase") {
        int index, count;
        cin >> index >> count;
        list.erase(index, count);
    } else if (choice == "get") {
        int index;
        cin >> index;
        cout << list.get(index) << endl;
    } else if (choice == "set") {
        int index;
        T value;
        cin >> index >> value;
        list.set(index, value);
    } else if (choice == "size") {
        cout << list.getSize() << endl;
    } else {
        list.print();
    }
}

int main() {

    LinkedList<int> list;

    int q;
    cin >> q;
    string choice;
    for (int i = 0; i < q; i++) {
        cin >> choice;
        Choice(list, choice);
    }

    return 0;
}
```

Завдання №4 Epic 6 Task 6 - Practice# programming: Class Practice Task

```cpp
#include<iostream>
#include<string>
using namespace std;

//Linked list
struct Node
{
    int data;
    Node* next;
};

//Додавання елемента в кінець
Node* push_back(Node* head, const int& element)
{
    Node* new_node = new Node;
    new_node->data = element;
    new_node->next = nullptr;

    if (head == nullptr)
    {
        head = new_node;
    }
    else
    {
        Node* current = head;
        while (current->next != nullptr)
        {
            current = current->next;
        }

        current->next = new_node;
    }

    return head;
}
```

```cpp
//Виведення списку
void print_list(Node* head)
{
    if (head == nullptr)
    {
        cout<<"List is empty"<<endl;
    }
    else
    {
        Node* current = head;
        while (current != nullptr)
        {
            cout<<current->data<<" ";
            current = current->next;
        }
        cout<<endl;
    }
}

//Реверс списку
Node* reverse_list(Node* head)
{
    if (head == nullptr)
    {
        cout<<"List is empty"<<endl;
        return 0;
    }
    else
    {
        Node* prev = nullptr;
        Node* current = head;
        Node* next = nullptr;

        while (current != nullptr)
        {
            next = current->next;
            current->next = prev;
            prev = current;
            current = next;
        }

        return prev;
    }
}
```

```cpp
//Порівняння на рівність двох списків
bool compare(Node* head_1, Node* head_2)
{
    Node* current_1 = head_1;
    Node* current_2 = head_2;

    while ((current_1 != nullptr) && (current_2 != nullptr))
    {
        if (current_1->data != current_2->data)
        {
            return false;
        }
        current_1 = current_1->next;
        current_2 = current_2->next;
    }

    if ((current_1 != nullptr) || (current_2 != nullptr))
    {
        return false;
    }
    else return true;
}

//Додавання двох великих чисел
Node* add_two_numbers(Node* n1, Node* n2)
{
    Node* current_1 = n1;
    Node* current_2 = n2;
    Node* sum = nullptr;
    int r = 0;
    int s = 0;
    while (current_2 != nullptr)
    {
        s = current_1->data + current_2->data + r;

        if (s > 9)
        {
            sum = push_back(sum, s % 10);
            r = s / 10;
        }
        else
        {
            sum = push_back(sum, s);
            r = 0;
        }

        current_1 = current_1->next;
        current_2 = current_2->next;
    }
    if (current_1 != nullptr)
    {
        while (current_1 != nullptr)
        {
            s = current_1->data + r;

            if (s > 9)
            {
                sum = push_back(sum, s % 10);
                r = s / 10;
            }
            else
            {
                sum = push_back(sum, s);
                r = 0;
            }

            current_1 = current_1->next;
        }
    }
    else if (r != 0)
    {
        sum = push_back(sum, r);
    }

    return sum;
}

//Виведення числа
void print_number(Node* head)
{
    if (head == nullptr)
    {
        cout<<"List is empty"<<endl;
    }
    else
    {
        Node* current = head;
        while (current != nullptr)
        {
            cout<<current->data;
            current = current->next;
        }
        cout<<endl;
    }
}
//Linked list

```

```cpp
//Binary tree
struct tree_node
{
    int data;
    tree_node* left;
    tree_node* right;

    tree_node(int value) : data(value), left(nullptr), right(nullptr){}
};



//Функція для вставлення елемента в дерево
tree_node* insert(tree_node* node, int value)
{
    if (node == nullptr)
    {
        return new tree_node(value);
    }
    else
    {
        if (value < node->data)
        {
            node->left = insert(node->left, value);
        }
        else if (value > node->data)
        {
            node->right = insert(node->right, value);
        }

        return node;
    }

}

//Віддзеркалення дерева
tree_node* create_mirror_flip(tree_node* node)
{

    if (node == nullptr)
    {
        return nullptr;
    }

    tree_node* new_node = new tree_node(node->data);

    new_node->right = create_mirror_flip(node->left);
    new_node->left = create_mirror_flip(node->right);

    return new_node;
}
```

```cpp
//Сума підвузлів
tree_node* tree_sum (tree_node* node)
{

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

    tree_sum(node->left);
    tree_sum(node->right);

    node->data = 0;
    if (node->right != nullptr)
    {
       node->data += node->right->data;
    }
    if (node->left != nullptr)
    {
       node->data += node->left->data;
    }

    return node;
}

//Виведення дерева
void print_tree(tree_node* node)
{
    if (node == nullptr)
    {
        return;
    }

    cout<<node->data<<" ";
    print_tree(node->left);
    print_tree(node->right);

}
//Binary tree
```

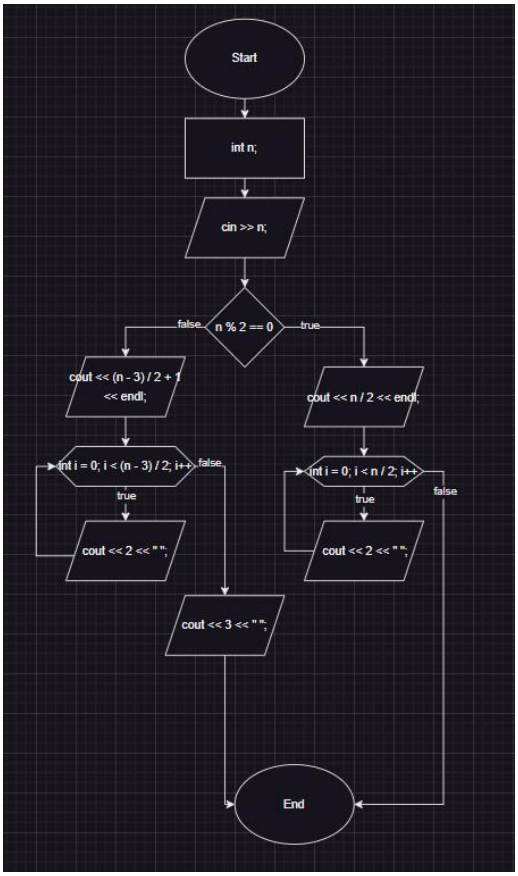
```cpp
int main()
{
    //task_1
    Node* head = nullptr;

    for (int i = 0; i < 10; i++)
    {
        head = push_back(head, i);
    }

    cout<<"Starting list: ";
    print_list(head);

    Node* new_head = reverse_list(head);
    cout<<"Reversed list: ";
    print_list(new_head);

    //task 2
    Node* head_1 = nullptr;
    Node* head_2 = nullptr;

    head_1 = push_back(head_1, 5);
    head_1 = push_back(head_1, 6);
    head_1 = push_back(head_1, 5);
    head_1 = push_back(head_1, 7);
    head_1 = push_back(head_1, 8);

    head_2 = push_back(head_2, 5);
    head_2 = push_back(head_2, 6);
    head_2 = push_back(head_2, 4);

    if (compare(head_1, head_2))
    {
        cout<<"Lists are equal"<<endl;
    }
    else cout<<"Lists aren`t equal"<<endl;

    //task 3
    string num_1, num_2, box;
    cout<<"Enter first number: ";
    cin>>num_1;
    cout<<"Enter second number: ";
    cin>>num_2;

    if (num_2.length() > num_1.length())
    {
        box = num_1;
        num_1 = num_2;
        num_2 = box;
    }
```

```cpp
320     Node* n1 = nullptr;
321     Node* n2 = nullptr;
322
323     for (int i = num_1.length() - 1; i >= 0; i--)
324     {
325         n1 = push_back(n1, (int)num_1[i] - 48);
326     }
327     for (int i = num_2.length() - 1; i >= 0; i--)
328     {
329         n2 = push_back(n2, (int)num_2[i] - 48);
330     }
331
332     Node* sum;
333     sum = add_two_numbers(n1, n2);
334
335     Node* new_sum = reverse_list(sum);
336     cout<<num_1<<" + "<<num_2<<" = ";
337     print_number(new_sum);
338
339     //task 4
340     tree_node* root = nullptr;
341
342     root = insert(root, 5);
343     root = insert(root, 3);
344     root = insert(root, 6);
345     root = insert(root, 2);
346     root = insert(root, 10);
347
348
349     tree_node* new_root;
350     new_root = create_mirror_flip(root);
351
352     cout<<"First tree: ";
353     print_tree(root);
354
355     cout<<"\nMirrored tree: ";
356     print_tree(new_root);
357
358     root = tree_sum(root);
359     cout<<"\nSum tree: ";
360     print_tree(root);
361
362     return 0;
363 }
```

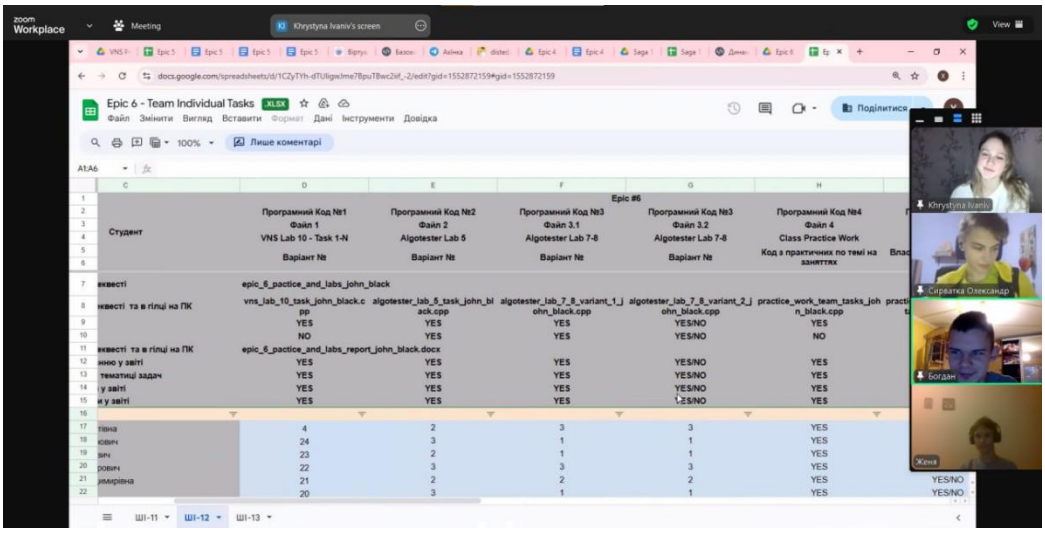
Завдання №5 Epic 6 Task 7 - Practice# programming: Self Practice Task

```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;

    if (n % 2 == 0) {
        // n парне: максимум груп розміром 2
        cout << n / 2 << endl;
        for (int i = 0; i < n / 2; i++) {
            cout << 2 << " ";
        }
    } else {
        // n непарне: одна група з 3, решта — з 2
        cout << (n - 3) / 2 + 1 << endl; // Одна група з 3 і решта з 2
        for (int i = 0; i < (n - 3) / 2; i++) {
            cout << 2 << " ";
        }
        cout << 3 << " ";

    }

    return 0;
}
```

Flowchart:



Робота в команді



Pull request: https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/619

Висновок: під час виконання лабораторної роботи я навчився краще працювати і розуміти динамічні структури. Навчився писати власні алгоритми для роботи з такими структурами.